Universitat de Lleida
Escola Politècnica Superior
Enginyeria Tècnica en Informàtica de Sistemes

Treball de final de carrera

Processat de Senyals d'Acceleració i Metodologia d'Ajust de Zero

Autor: Albert Agraz Sánchez
Director: Francisco Clariá Sancho
Gener de 2010

Abstract

The aim of this project is to design a system to capture the acceleration of an autonomous mobile device in the three dimensions based on an integrated circuit sensor of acceleration.

Using cheap microcontrollers we try to apply some changes to the signal in order to improve it, remove noise and obtain better precision.

The development of the application that implements such improvements is done using Software Engineering techniques, in order to produce a stable, efficient and ordered application that can easily be introduced to the system.

Índex de Continguts

Índex de Figures

Índex de Codi

1.Introducció

En aquest capítol s'explicaran quins són els objectius d'aquest projecte, quin és l'estat actual de la tecnologia que ens ha portat a desenvolupar aquest projecte, quina és la proposta o idea bàsica de solució i finalment quina serà l'estructura de la memòria.

1.1. Objectius

L'objectiu principal d'aquest projecte és l'estudi i el disseny d'un sistema mesurador en temps real d'acceleració en tres eixos ortogonals basat en un circuit integrat que proporciona senyals d'acceleració per tal d'adaptar-lo a un dispositiu autònom.

Essencialment, es tracta del disseny d'un sistema que ens permeti obtenir l'acceleració en els tres eixos d'un dispositiu que es mogui en tres dimensions.

Més enllà de d'aquest objectiu, trobem objectius menors que hauran de ser assolits per tal de poder satisfer l'objectiu principal:

- Realitzar un estudi de mercat i escollir el millor equipament per a desenvolupar el nostre projecte.
- Estudiar el sensor d'acceleració i aplicar-hi millores per tal d'obtenir una senyal més acurada.
- Dissenyar un procediment que ens apliqui les millores a les dades obtingudes del sensor en temps real.
- Provar aquestes millores i determinar si són o no eficaces per al sensor estudiat.
- Avaluar els diferents protocols de comunicació entre microcontroladors i escollir el més adient al nostre projecte.

- Dissenyar una aplicació, seguint els patrons de l'Enginyeria del Software, capaç d'obtenir les dades del sensor i aplicar-hi el procediment adient per a millorar la seva qualitat.
- Aprendre el funcionament general d'un microcontrolador amb gran capacitat de còmput.
- Dominar l'entorn de programació de microprocessadors CodeWarrior.
- Codificar, de forma ordenada i eficient una aplicació que compleixi amb el disseny concebut.

Al llarg d'aquest projecte intentarem assolir els objectius aquí plantejats, així com altres reptes que segur se'ns aniran plantejant, però que, al no estar dins dels objectius inicials, han quedat reflectits com annexes o bé anotacions.

1.2. Estat Tecnològic Actual

En aquest apartat, intentem explicar com està el mercat i la investigació actual respecte als sensors d'acceleració tridimensionals. Intentarem resumir breument què hem trobat.

Actualment trobem, cada vegada més, sensors tridimensionals d'acceleració especialitzats en detectar vibracions. Són sensors de baixes prestacions però que detecten sotracs i moviments molt bruscos, però que no estan preparats per a detectar acceleracions constants en el temps, produïdes, per exemple, pel continu augment de velocitat d'un dispositiu.

Aquests sensors de baixes prestacions estan destinats al mercat de l'entreteniment. Són el puntal dels dispositius que

reconeixen moviments i permeten interactuar amb major llibertat amb videojocs, telèfons mòbils, etc.

També trobem dispositius especialitzats en la detecció de la caiguda lliure. Son dispositius que basats en un acceleròmetre, activen una sortida quan detecten que estan sotmesos a l'acceleració de la gravetat. Aquests estan més enfocats a protegir unitats de disc dur, etc.

Finalment, resulta més costós trobar dispositius orientats a la detecció de l'acceleració en qualsevol circumstància, no només en un xoc o una caiguda. Aquests dispositius, a més de ser força més cars, no són gaire precisos. O bé, són precisos però cometen un error mínim que condiciona molt el seu ús.

En definitiva, en el mercat actual no trobem un producte que sigui, sense modificació, suficientment precís i lliure d'error per a detectar l'acceleració tridimensional d'un dispositiu mòbil autònom que es basarà en aquesta magnitud per a realitzar càlculs de trajectòria.

1.3. Proposta de Solució

Coneixent els nostres objectius i sent conscients del que ofereix el mercat, la nostra proposta de solució ha de definir les línies principals d'un projecte que pugui satisfer els objectius amb els recursos del mercat i el nostre treball.

La nostra proposta es basa en el fet que el mercat sí que ofereix sensors d'acceleració precisos però que aquests presenten uns errors sota unes circumstàncies determinades. Basats en l'esperança que aquests errors poden ser estudiats i corregits

mitjançant mètodes de tractament del senyal, definirem un sistema que s'encarregui d'aquests errors i presenti uns senyals d'acceleració lliure nets.

El centre del nostre sistema serà, per tant, el sensor d'acceleració. Llegint aquesta acceleració situarem un microcontrolador que serà l'encarregat de fer el tractament del senyal. Escollim un microcontrolador per les seves grans capacitats de còmput i d'interacció amb la resta de components d'un sistema.

Aquest microcontrolador haurà d'enviar els seus resultats a través d'un canal concret cap a un receptor que serà qui processarà aquestes dades, en definitiva, qui les ha demanat.

El sistema, a més, haurà de ser ràpid i no podrà ocupar gaire, per a oferir senyal en temps real i no afectar al dispositiu autònom al que serveix. Un esquema general del sistema podria ser aquest:

Figura 1.1: Proposta de Solució

1.4. Estructura de la memòria

La memòria s'ha estructurat segons el procés d'estudi que s'ha realitzat. Es va començar amb la tria dels components, després s'ha fet un estudi del senyal, del microcontrolador i del bus. Finalment s'ha dissenyat i implementat l'aplicació:

- En el capítol 1 s'introdueix el projecte i es parla dels objectius i les possibilitats que es tenen per a assolir-los.
- En el capítol 2 s'estudia el mercat en busca dels components més adequats per al nostre sistema. S'avalua tant cost, com facilitat de treball, estandardització, etc.
- En el capítol 3 es realitza un estudi de tot el tractament del senyal que es pot aplicar al sensor escollit en el capítol 2. Es dissenyen i es comproven uns procediments que es creuen milloraran el comportament del sistema.
- En el capítol 4 es fa una recerca sobre els diferents mòduls del microcontrolador que s'empraran en aquest projecte. Es remarquen configuracions, modes d'ús i funcionament intern. S'estableix una base per al futur desenvolupament.
- En el capítol 5 es presenta l'entorn de programació que facilita el fabricant del microcontrolador. Es tracten aquells aspectes comuns amb altres entorns, així com els particulars del desenvolupament d'aplicacions orientades a microprocessadors.
- En el capítol 6 es desenvolupa l'aplicació software que implementa el tractament del senyal. Primer es tria un mètode de desenvolupament i, seguint les prescripcions d'aquest, es detallen tots els passos realitzats.
- En el capítol 7 es parla d'una valoració que han de tenir tots els projectes, la valoració de costos. Al tractar-se d'un projecte d'investigació, es detallen les hores invertides i el material utilitzat, no el cost econòmic.
- En el capítol 8 s'exposen les conclusions a les que s'ha arribat, ja sigui satisfactòries com no satisfactòries, i es detalla la feina que encara queda per fer.
- En el capítol 9 s'inclouen referències a la bibliografia consultada per a la realització d'aquest projecte, per a que el

lector interessat pugui ampliar coneixements o comprovar deduccions.

- L'annex A inclou el codi font de l'aplicació en el moment de la redacció d'aquesta memòria.
- L'annex B ens mostra el procediment de calibrat d'un sensor de temperatura analògic. Aquest procediment, no considerat com un objectiu, s'ha inclòs en aquesta memòria per la seva rellevància.
- L'annex C estableix les bases per a un protocol I^2C que comuniqui diversos sistemes de captació de dades amb un únic gestor.
- L'annex D mostra una petita rutina per a enviar dades compostes a través d'un protocol que només permet transmetre *byte* a *byte*.

2.Tria dels Components

Un cop hem vist la proposta de solució que oferim per al nostre problema, i ens ha quedat clar l'esquema general del sistema, és hora d'escollir els components que formaran aquest sistema.

Realitzar una bona tria és vital per al bon desenvolupament del projecte, ja que segons la qualitat i les característiques dels components, podrem fer les coses d'una forma, d'una altra o bé no realitzar-les.

Per a triar els components farem un petit estudi de mercat i escollirem, segons les alternatives trobades, l'opció que millor s'adeqüi a les nostres necessitats. No valorarem tan sols les característiques tècniques, sinó també altres factors que poden resultar determinants i no semblar importants a primer cop d'ull com, per exemple, la facilitat d'obtenció dels components, o bé la plataforma de desenvolupament associada.

El primer que triarem és el sensor d'acceleració. Aquesta tria ens condicionarà completament la del microcontrolador, ja que segons com sigui el mètode de transferència de dades de l'acceleròmetre, necessitem un microprocessador amb unes característiques o unes altres.

Amb el sensor ja escollit, haurem de pensar quin és el microcontrolador més adequat per al sistema. Com hem comentat, aquesta tria estarà completament condicionada a la forma de la sortida del sensor. Caldrà, doncs, escollir un processador apte per a capturar la senyal i tractar-la.

Finalment, i tenint en compte que el nostre acceleròmetre tridimensional formarà part d'un sistema força més gran i complex, hem d'escollir un bus o tecnologia que ens permeti comunicar-nos amb l'exterior. La tria d'aquest bus haurà d'estar justificada segons quotes de mercat i estandardització.

Per tal de fer més senzilla la lectura d'aquest document hem dividit aquest apartat en 3 de més petits, un per cada tria que haurem de realitzar. El lector trobarà útil refrescar l'esquema global de la proposta de solució del sistema (figura 1.1) explicat en el capítol anterior.

2.1. Sensor d'Acceleració

El sensor d'acceleració és el nucli del nostre sistema. Sobre aquest sensor desenvoluparem la resta del nostre disseny, per tant, l'elecció d'aquest component és la decisió més important que prendrem en aquest capítol.

Abans de centrar-nos en les característiques que ha de tenir el sensor que requerim, és important que ens familiaritzem amb el funcionament intern d'un acceleròmetre per tal d'entendre en què ens hem de fixar.

2.1.1. Principi de Funcionament

Mesurar l'acceleració és una cosa que pot sembla molt abstracta en un primer moment. Per començar, l'acceleració és una magnitud relativa al punt de referència que establim.

Trobar l'acceleració d'un cos, en definitiva, no és senzill. Una opció que se'ns acudeix és aprofitar el fet que una part lliure sobre un cos accelerat no està sotmesa a la mateixa acceleració. Un exemple d'aquest fet és si situem una pilota sobre una superfície plana. Si accelerem la superfície, la pilota no accelera de la mateixa forma, i si el fregament no és suficient, caurà per l'altre costat. Igualment, si apliquem una acceleració negativa (frenem) a la mateixa superfície en moviment, la pilota sortirà disparada endavant.

Una aplicació d'aquest fenomen la trobem en els cinturons de seguretat dels vehicles. El cinturó només es bloqueja en cas de

detectar una acceleració molt forta. Per això utilitza una petita bola imantada. Aquesta bola es troba magnèticament atreta per una petita planxa metal·litzada. Si sotmetem el vehicle a una acceleració suficientment intensa per a superar la força de l'imant, la bola es separà de la planxa i el cinturó es bloquejarà.

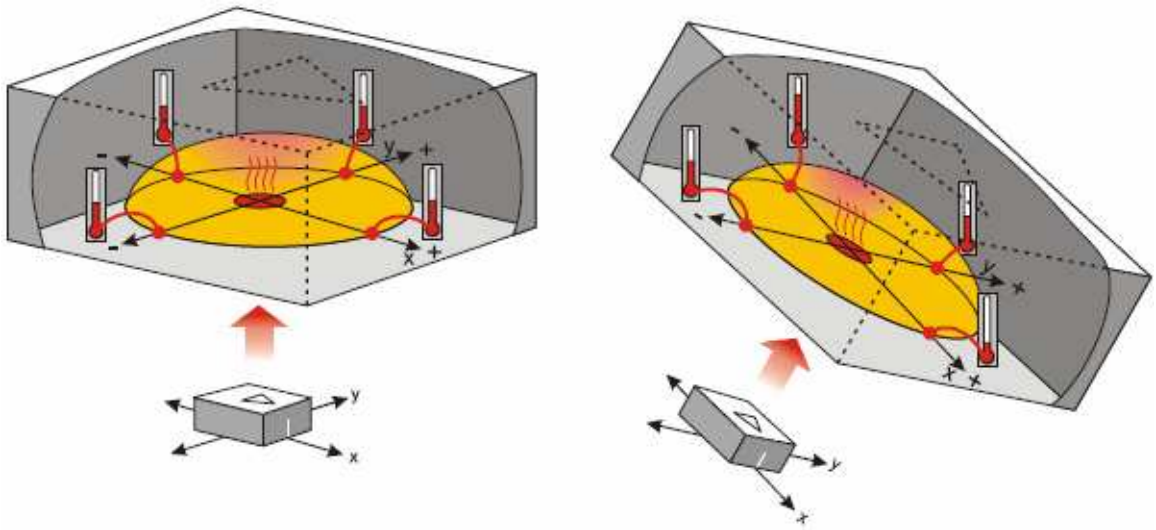

Figura 2.1: Principi de Funcionament de l'Acceleròmetre

Aplicar aquest principi a un xip no és senzill. De fet, els xips no tenen parts mòbils, per això són més fiables. Així doncs, el que s'aplica és el mateix principi però en una bombolla de gas.

Imaginem-nos una bombolla d'un gas calent dins d'una petita cambra quadrada. A cada paret de la cambra situarem un termistor. En un principi, amb el sensor en repòs, la bombolla es trobarà al centre de l'habitacle, a la mateixa distància de tots els termistors. En aquest moment els 4 termòmetres registraran la mateixa temperatura.

Si accelerem, la bombolla no accelerarà igual que el sensor i, per tant, s'aproximarà al termistor de la paret oposada al sentit de l'acceleració, igual que la pilota es "desplaçava" cap al costat oposat al moviment de la superfície.

No és difícil advertir, doncs, que al mesurar l'acceleració a través de la diferència de temperatures enregistrada per uns termistors, el funcionament global del sensor dependrà en certa mesura de la temperatura a la que es trobi el xip.

Finalment, si afegim termistors al terra i al sostre de la cambra podrem tenir un sensor d'acceleració en els 3 eixos.

2.1.2. Característiques Bàsiques

Ara que ja coneixem el principi bàsic de funcionament dels acceleròmetres, podem començar a parlar de les especificacions mínimes que ha de tenir el nostre sensor.

El primer del que parlarem és sobre el rang de treball i la precisió. Idealment no són paràmetres que hagin d'estar relacionats, però a la pràctica aquests dos elements depenen força un de l'altre: a major precisió menor rang i a major rang, menor precisió.

Segons les estimacions realitzades per a la nostra aplicació, acceleròmetre tridimensional amb aplicacions en la aeronàutica, hem estimat adequat un rang de treball de ±3g[1]. Aquesta magnitud és considerable, tenint en compte que l'acceleració de la gravetat és força intensa.

En quant a la precisió buscarem el sensor més precís que puguem i que també compleixi les altres condicions. Aquesta magnitud preferim no fixar-la, ja que com més precís sigui, millor.

1 En aquest document quantificarem les acceleracions en funció de l'acceleració de la gravetat (g) i no en m/s2.

De totes formes, al voltant d'una centèsima part de g és un valor molt adequat.

Una característica que resulta imprescindible per al nostre treball i que, en canvi, no contemplen la majoria de fabricants és el termòmetre del xip. Com hem vist en l'apartat anterior, el sensor d'acceleració es basa en la temperatura, i la temperatura global del dispositiu pot servir com a agent corrector dels errors comesos pel dispositiu. Si el que volem és millorar la senyal de sortida, haurem de poder capturar la temperatura a la que es troba el sensor. Per això hi ha d'haver un termòmetre incorporat al xip.

També hem de tractar la possibilitat de tenir diversos eixos en un mateix xip. Si escollim un sensor que només ens retorni l'acceleració en dos eixos, haurem de ficar dues unitats d'aquest model en el nostre sistema, mentre que si directament escollim un sensor que tingui 3 eixos, amb una unitat serà suficient.
Aquesta característica resulta adequada però no serà determinant, ja que si no podem trobar un sensor de 3 eixos que compleixi les altres característiques, però sí que existeix amb 2 eixos, només haurem d'adquirir 2 unitats d'aquest últim.

En definitiva, a l'hora d'escollir el sensor d'acceleració adequat per al nostre projecte, tindrem en compte aquests requisits:

- Rang de treball de ±3g
- Precisió de $g \cdot 10^{-2}$
- Sensor de temperatura
- 3 eixos

D'on aquesta última característica no resultarà determinant a l'hora d'escollir un sensor, però sí recomanable.

2.1.3. Estudi de Mercat

Un cop hem definit les característiques que ha de tenir el nostre sensor d'acceleració, podem cercar en el mercat les diferents alternatives que tenim per tal d'escollir la que s'adapti millor a les nostres necessitats.

El primer que hem de fer és realitzar una cerca dels principals fabricants de sensors d'acceleració MEMS[1]. Seran els catàlegs d'aquests fabricants els que ens serviran per a saber si els productes actualment al mercat poden satisfer els nostres requisits. Els fabricants que hem decidit consultar són els següents:

- Analog Devices
- Freescale
- MEMSIC

Analog Devices

El primer fabricant, Analog Devices, treballa amb una gran quantitat d'acceleròmetres de diferents característiques. Els classifica segons el rang de treball i la precisió: per a grans acceleracions o per a petites acceleracions. Els nostres requisits formen part de les "petites acceleracions".

Dins d'aquesta família trobem dispositius amb diverses precisions, més d'un compleix els nostres requisits de precisió. El problema arriba amb el sensor de temperatura. Gairebé cap dels xips produïts per aquesta empresa incorpora un termòmetre per a realitzar correccions, i d'incorporar-lo l'utilitza per a fer correccions internes, i l'usuari no té accés a aquesta senyal.

[1] MicroElectroMechanical Systems, en les seves sigles en anglès.

Per aquest motiu els dispositius d'Analog Devices queden descartats directament. Ja que sense una temperatura amb la que realitzar correccions, difícilment podrem millorar la senyal de sortida que, segur, contindrà errors.

Freescale

El segon fabricant, Freescale, és una filial de Motorola que implementa diverses solucions hardware, entre altres, microcontroladors. En el camp dels acceleradors disposen de molts pocs productes, i amb unes característiques força inferiors a les d'Analog Devices.

Tot i tenir aproximadament els mateixos rangs de treball, no disposen de la mateixa precisió, i sembla que aquests dispositius són més aviat per a detectar moviment que per a quantificar-lo.
Tampoc disposen d'una senyal de temperatura per a condicionar les dades externament, per tant també descartarem aquesta opció.

MEMSIC

Finalment, l'últim fabricant que queda per comprovar és MEMSIC. Aquesta empresa disposa d'una gran quantitat de sensors d'acceleració, de diferents rangs, amb diferents precisions, etc.
Entre aquesta varietat, trobem diversos sensors amb un rang de treball adequat (3g) i una precisió superior a la requerida. En aquest cas, però, el fabricant no ha deixat de banda la possibilitat de millora del senyal incorporant un termòmetre al xip.

L'última característica que queda per comprovar és la quantitat d'eixos amb la que treballa. Desgraciadament només trobem unitats de dos eixos que compleixin les altres

característiques, per tant haurem de treballar amb dues unitats del sensor d'acceleració i ometre un dels eixos d'un dels xips.
Concretament, l'acceleració dels eixos, és a dir, la senyal d'acceleració, s'envia a l'exterior codificada en un tren de polsos, és a dir, en cada pols del tren, segons la seva amplitud, codifiquem una acceleració. Per tant, tindrem 2 trens de polsos per cada sensor i a més una senyal analògica en voltatge que ens indica la temperatura del sensor.

Aquest model és el **MXD2125G**, tot i que ens planteja un inconvenient. El format d'encapsulat és una pastilla amb les connexions directes, sense pins per a soldar. Aquest sensor està pensat per a ser soldat directament sobre la placa. Per evitar haver de treballar amb aquest tipus de soldadura, que només es pot realitzar amb braços de soldadura, emprarem una solució de Parallax. Aquesta empresa adquireix els sensors MXD2125G de MEMSIC i els solda a una petita PBC amb els pins accessibles per a poder treballar amb ells i soldar-los de forma tradicional a un altra placa (figura 2.2).

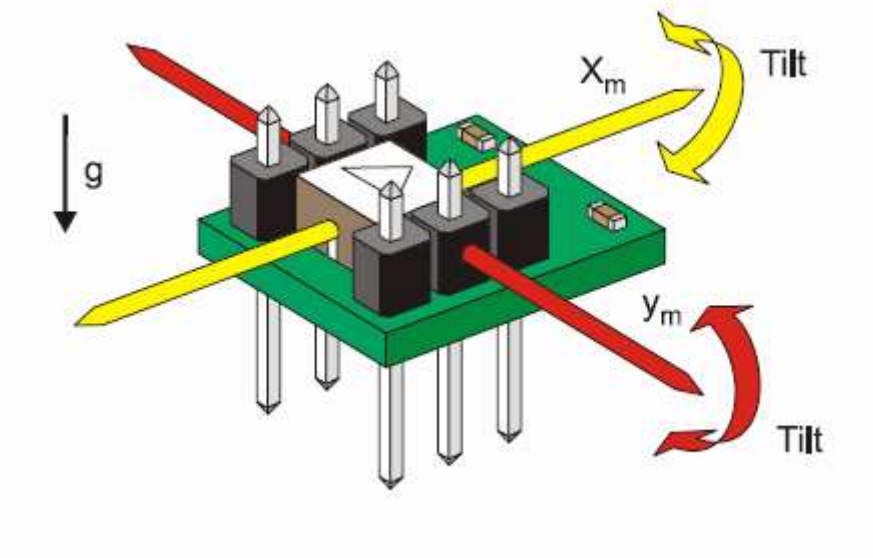

Figura 2.2: Esquema de Muntatge del Sensor

També creiem convenient analitzar breument el diagrama de blocs del sensor. Ara és un bon moment per a observar com funciona, ja que més endavant ens centrarem únicament en el senyal que treu.

Com podem veure en la figura 2.3, aquest sensor està format per diversos components, que estudiarem breument:

- **Oscil·lador Intern** (*Internal Oscillator*): és l'oscil·lador que generarà el tren de polsos de la sortida.
- **Sensor de Temperatura**: és el termòmetre que ens indicarà la temperatura del xip, te una sortida analògica i també és utilitzat internament.
- **Referència de Voltatge**: es tracta de la unitat que estableix el voltatge de referència a partir de l'alimentació.
- **Control d'Escalfament** (*Heater Control*): és el component que escalfa la bombolla de gas, a una temperatura determinada, condicionada per la temperatura global del xip.
- **Auto-Test Continu**: una unitat que constantment compara el valor que s'envia a l'exterior amb el valor intern.
- **Termistors**: són els sensors que detecten la temperatura en la cambra on es troba la bombolla.
- **Ajust de Offset i Guany** (*Factory Adjust Offset & Gain*): es tracta de l'ajust que realitza el sensor a la senyal abans d'enviar-la a l'exterior. Aquest ajust no te en compte la temperatura, sinó que treballa com si sempre estigués a una temperatura òptima de 25ºC. Aplica els canvis (sempre constants) a través d'un amplificador operacional per cada eix.

- **Filtre de Pas Baix** (LPF[1]): el sensor, abans d'enviar la senyal, li aplica un filtre pas baix per tal d'eliminar soroll de fons. Aquest filtre és força dolent, per això caldrà tornar a filtrar la senyal.
- **A/D**: l'últim que fa el sensor és transformar la senyal d'analògic a digital per tal de poder codificar-la en el tren de polsos. Nosaltres ja no l'haurem de tornar a analògic, sinó que directament podrem capturar el tren de polsos.

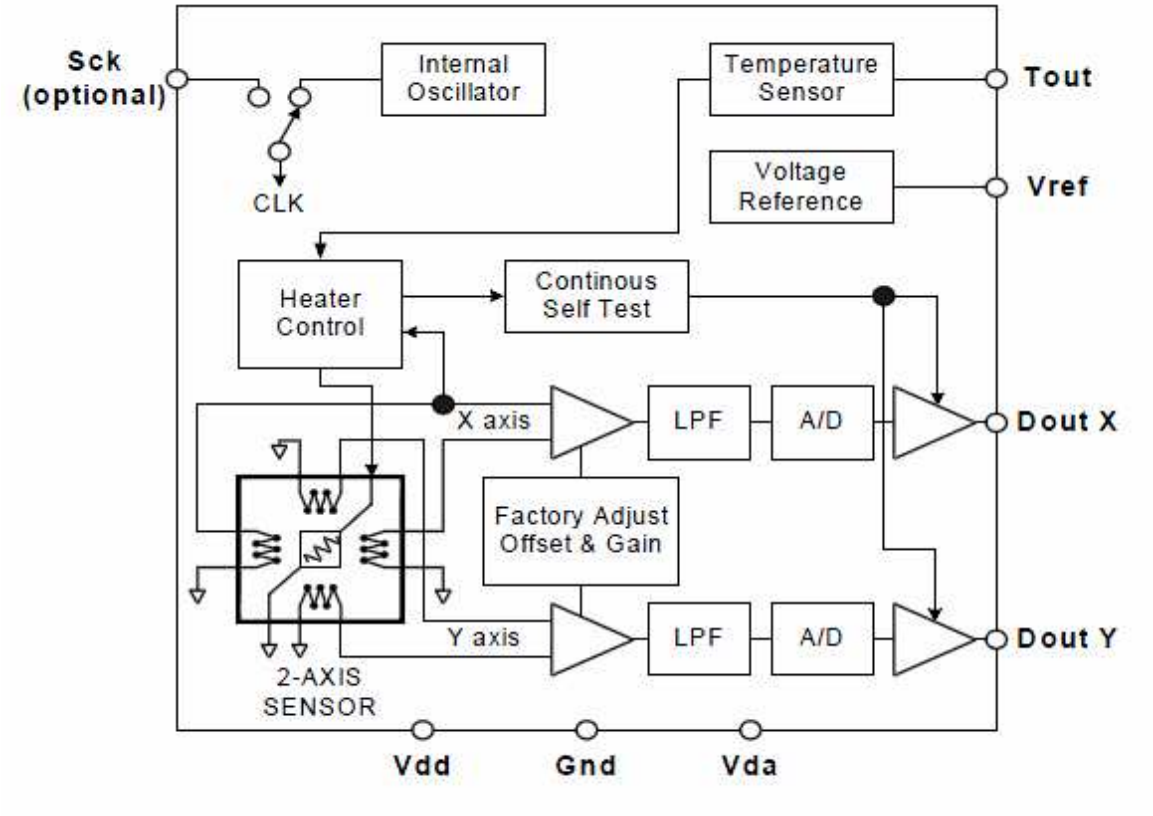

Figura 2.3: Diagrama de Blocs del Sensor

Definitivament ja tenim seleccionat el sensor d'acceleració. Després de la cerca i el petit estudi dels seus components, podem afirmar que aquest sensor compleix amb escreix les nostres expectatives. Superada la problemàtica de l'encapsulat, podem dir que el **MXD2125G** serà el nostre sensor d'acceleració.

[1] Low Pass Filter, en les seves sigles en anglès.

2.2. Microcontrolador

En el mercat dels microcontroladors hi ha molts fabricants que elaboren les seves pròpies solucions. Altres fabricants implementen les arquitectures dissenyades per altres. En tots els casos, quan arribem a un nivell de complexitat de 32 bits, les coses canvien. Són menys els que ofereixen un producte final, practicable i amb característiques avançades.

Nosaltres no estem buscant només un microprocessador que ens permeti una gran capacitat de còmput i una interacció amb el món exterior, sinó que el que aquí analitzem és un entorn de desenvolupament complet, amb tot el necessari per a produir un producte final.

D'entre tots els fabricants, hem escollit els dos que ofereixen productes d'alta complexitat però amb eines que els fan accessibles als petits laboratoris d'investigació: Texas Instruments i Motorola, a través de la seva filial Freescale.

D'entre les diferents famílies que ofereix cada fabricant, hem buscat aquella que més s'adequa als nostres requeriments a priori:

- Instruccions de 32 bits.
- Capacitat de còmput elevada (més de 60 MHz).
- Memòria RAM capaç de treballar amb força dades (mín. 96KB)
- Memòria Flash suficient per aplicacions grans (aprox. 128KB).
- Possibilitat de capturar trens de polsos.
- Unitat Analògica-Digital

Com veiem, un dels 5 requeriments principals, i excloent, és el de poder adquirir les dades que ens envia el sensor MXD2125G.

Emprant els catàlegs de productes dels dos principals fabricants, escollirem una família per cada fabricant que més s'apropi a les característiques sol·licitades. Diem que més s'apropi per que sempre podríem escollir les famílies amb unes característiques més avançades i segur que satisfarien els nostres requeriments, però el sobrecost seria molt important.
Les dues famílies més adequades són:

- *C2000 Delfine Family* de Texas Instruments
- *ColdFire V2 MCF521X* de Motorola

El que hem de fer ara és veure les característiques més rellevants de cada família i escollir la més adequada. Després escollirem el microcontrolador idoni d'entre la família triada.

2.2.1. C2000 Delfine (Texas Instruments)

La família C2000 de Texas Instruments és un grup de microcontroladors de prestacions considerables altament utilitzant en entorns empresarials.

Les principals característiques dels xips d'aquesta família que compleixen les nostres característiques, són:

- Instruccions de 32 bits.
- Freqüència de rellotge de 100-150MHz
- RAM: 52-68KB
- Flash: 128-512KB
- ADC: 16 canals de 12 bits
- 16-18 canals de PWM
- 4 temporitzadors
- 6 DMA
- Representació *floating-point*.
- 1 I^2C

- 3 UART
- 1 SPI
- 2 CAN
- 88 GPIO
- Cost: 13’85 – 21’00 USD (comprant-ne 1000)

En el cas de l’entorn de desenvolupament, Texas Instruments ofereix una petita targeta d’avaluació que es connecta per USB amb l’equip i pot realitzar la programació del microcontrolador. Aquesta targeta, de baix cost, no ofereix possibilitats d’entrada i sortida estàndard, com, per exemple, UART o SPI. Si es vol tenir aquestes facilitats en la targeta d’avaluació cal adquirir models més costosos.

En quant al desenvolupament del software, ens ofereix una *suite* de producció anomenada Code Composer Studio que ens permet realitzar totes les tasques relacionades amb el xip.
Podrem editar el nostre codi, construir la nostra aplicació, depurar el nostre codi, etc.
Una de les eines que mereixen una menció especial sobre aquesta aplicació és la possibilitat de desenvolupar codi i simular-lo sense la necessitat de disposar físicament del microcontrolador. El propi programari ofereix la possibilitat d’emular el xip per tal de facilitar el desenvolupament i les proves al programador.

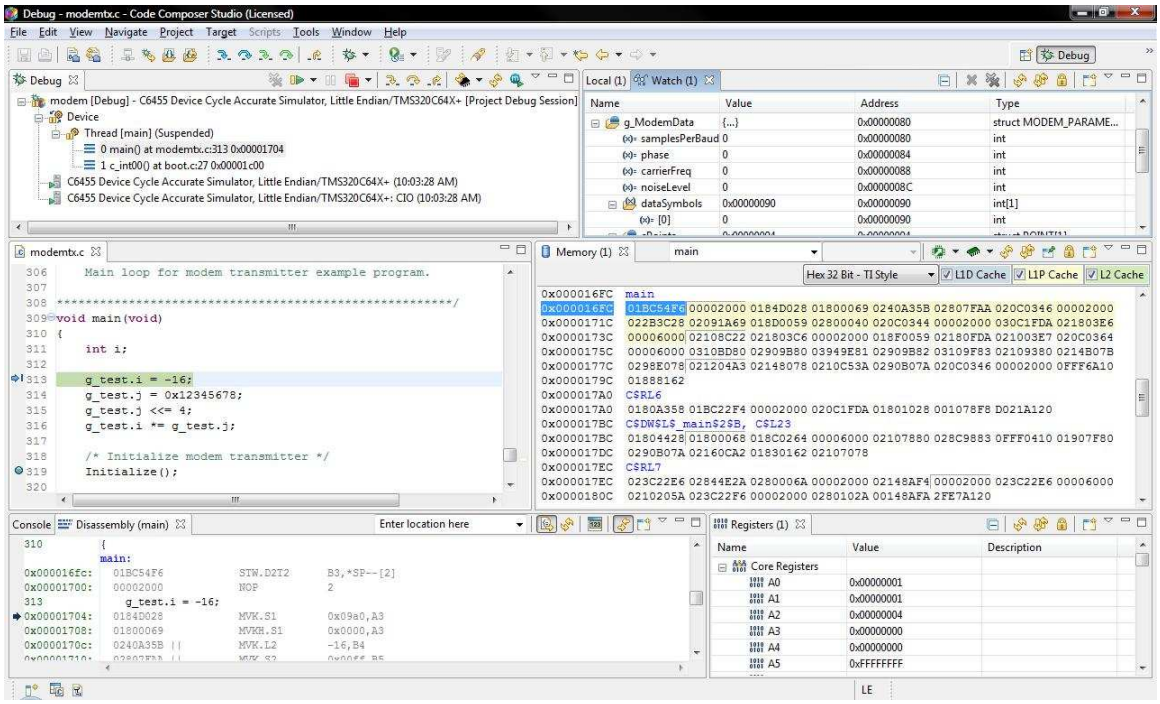

Figura 2.4: Entorn de Desenvolupament CCS

Aquesta eina està basada en Eclipse, per tant l'entorn serà molt estable (figura 2.4) i desenvolupat en base a programari lliure. L'aplicació de TI, però, és de pagament.

2.2.2. ColdFire V2 MCF521X

Dins de les famílies de microcontroladors de Motorola, la tecnologia de gamma més adequada a les nostres necessitats és la V2. Si observem el mapa de ruta (*roadmap*) dels microcontroladors de la branca ColdFire veurem com sobre les capacitats de memòria mínimes i amb les característiques necessàries trobem la família MCF521X.

Es tracta d'una família amb una potència de càlcul adequada per a poder aplicar filtres i amb unes possibilitats d'interactuar amb diversos tipus de senyals molt àmplies, a més de portar incorporats

diferents protocols de comunicació. Vegem aquestes característiques :

- Instruccions de 32 bits.
- Freqüència de rellotge de 60-80MHz
- MAC Module i HW Divider
- RAM: 16-64KB
- Flash: 128-512KB
- ADC: 8 canals de 12 bits
- 4 canals de PWM
- 4 temporitzadors de 32 bits amb DMA
- 4 DMA
- Representació floating-point.
- 1 I^2C
- 3 UART
- 1 QSPI
- FlexCAN 2.0 amb 16 *buffers* per a missatges
- 55 GPIO
- 2 Programmable Interrupt Timers
- Cost: 7’93 - 8’74 USD (comprant-ne 1000)

Com a entorn de desenvolupament, Freescale ofereix una targeta d’avaluació molt completa. Disposa dels 3 ports UART, i accés a tots els pins rellevants del xip, a través dels que podrem treballar amb tots els mòduls del microcontrolador. A més a més, la connexió amb aquesta targeta es realitza per USB, amb el que no hi haurà problemes de compatibilitat.
El cost d’aquesta targeta no supera els 300€.

En quant a l’aplicació. Motorola ofereix una eina abans desenvolupada per Metroworks: CodeWarrior. Aquesta eina integra tot el procés de tractament amb el microcontrolador. Des de la configuració inicial fins a la programació, oferint eines tant

necessàries com el depurador de programes. Val a dir que el cost d'aquesta eina és força elevat.

2.2.3. Tria de la Família i el Model

Un cop hem analitzat les dues famílies més adaptables a les nostres necessitats i els entorns que ens proporcionen, és el moment de prendre una decisió.

Ambdues famílies presenten unes característiques suficients per a desenvolupar la nostra aplicació, però els costos no són els mateixos. Si observem els preus per unitat, la família de Texas Instruments és un 50% més cara que la de Motorola i en teoria ens poden servir les dos.

L'entorn de desenvolupament també ens ha de servir per a marcar la nostra decisió. És veritat que ambdós entorns són molt estables i estan reconeguts dins de la indústria, però la targeta d'avaluació que ofereix Freescale és força millor que l'entorn de treball que ens proposa TI.

Finalment, el fet que disposem d'un major accés al programari i targeta d'avaluació de Freescale, farà que ens decidim per aquesta última solució.

Ara hem de triar un dels microcontroladors d'aquesta família. El primer que farem és buscar una comparativa de les característiques bàsiques de cada model (figura 2.5).

MCF521x Family				
Part Number	**Flash/ SRAM**	**Key Features**	**Package**	**Speed**
MCF5211	128 KB/ 16 KB	3 UARTs, I²C, QSPI, 16-bit, 32-bit, PWM timers, A/D	64-pin LQFP 81-ball MAPBGA	66 80
MCF5212	256 KB/ 32 KB	3 UARTs, I²C, QSPI, 16-bit, 32-bit, PWM timers, A/D	64-pin LQFP 81-ball MAPBGA	66 80
MCF5213	256 KB/ 32 KB	3 UARTs, I²C, QSPI, 16-bit, 32-bit, PWM timers, A/D, CAN	81-ball MAPBGA, 100-pin LQFP	80 80

Figura 2.5: Característiques bàsiques dels models MCF521X

Bàsicament, la diferència es troba en la quantitat de memòria, la possibilitat de treballar amb el bus CAN i la freqüència.
Per als nostres objectius, preferim treballar amb una memòria gran, per tant, descartarem el MCF5211. Dels dos models restants, disposem de versions a 80MHz, per tant aquest factor no serà determinant.
Sí que ho serà, per altra banda, la possibilitat de treballar amb el bus CAN. Donat que volem que el nostre sistema sigui altament interconnectable amb altres components, considerem més oportú apostar per un microcontrolador amb més capacitat de comunicació.

Així doncs, després d'analitzar les famílies, escollir-ne una, i comparar els diferents models de la família escollida, hem sigut capaços de determinar que el microcontrolador més adequat per a la nostra aplicació és el **MCF5213**.

2.3. Bus de Dades

Un cop ja hem escollit el sensor d'acceleració a utilitzar i el microcontrolador que llegirà les dades i les tractarà, només ens queda decidir quin bus i protocol utilitzarà el microprocessador per a comunicar-se amb la resta del sistema, és a dir, per enviar les dades tractades a qui les hagi de fer servir.

Aquesta tria es troba condicionada per dos factors. Per una banda tenim els requisits mínims que ha de complir per tal de poder fer la feina que necessitem.
Per altra banda, aquest bus ha de ser compatible amb el MCF5213, per tant, haurà de ser un dels que aquest microcontrolador te implementats.

Passarem ara a descriure breument els diversos bussos que ens ofereix el MCF5213, i al final d'aquest capítol triarem un d'ells per a ser la porta de comunicació amb l'exterior del nostre sistema:

- QSPI
- UART
- I^2C
- FlexCAN

2.3.1. Queued Serial Peripheral Interface (QSPI)

El QSPI és un protocol orientat a la comunicació amb components perifèrics del microcontrolador. Està orientat a la transmissió de paraules de 16 bits utilitzant 7 bits. Disposa d'uns bits de selecció per a escollir el component al que ens estem

adreçant. Podem veure un esquema general del mòdul en la figura 2.6.

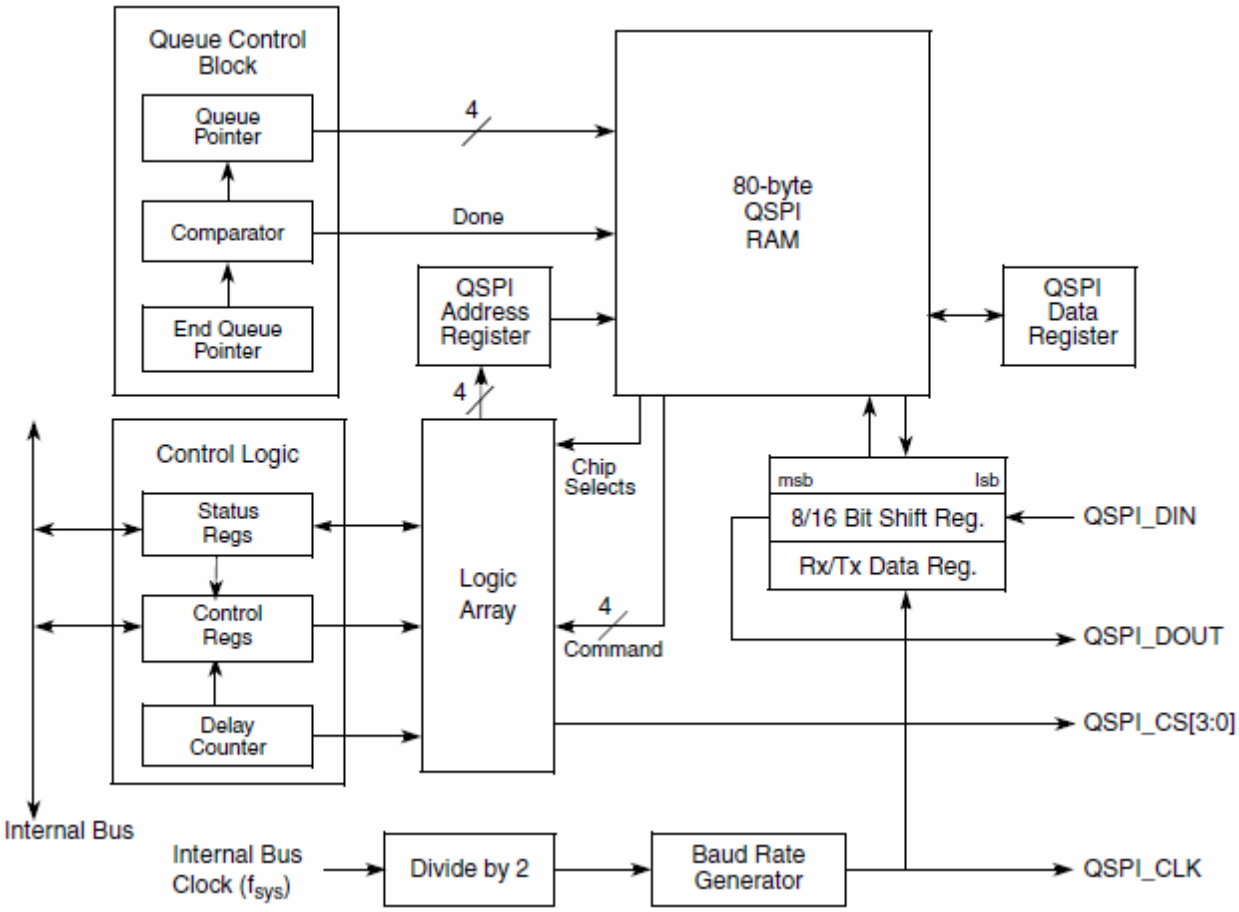

Figura 2.6: Diagrama de Blocs QSPI

Les característiques principals d'aquest mòdul com a part del MCF5213 són les següents:

- Cua programable fins a 16 paquets
- Suport de transferències de 8 a 16 bits de longitud
- Possibilitat de selecció de 16 dispositius
- Transferències des de 129'4Kbps a 16'6Mbps
- Retards programables entre transferències
- Rellotge QSPI configurable

Com podem veure, es tracta d'un protocol força complet, que ens permet treballar a velocitats molt adequades i amb un nombre de perifèrics considerable.

Un dels inconvenients que presenta aquest bus és la complexitat de funcionament que te. Així com podem trobar bussos que amb una lleugera configuració transmeten dades, el QSPI requereix d'una configuració força feixuga.

2.3.2. UART

El bus UART és un bus molt utilitzat a nivell comercial. Tot i que per aquestes sigles no se l'acostuma a conèixer, es tracta del port sèrie del que disposen molts ordinadors.

L'avantatja d'aquest bus, a part de la seva àmplia utilització, és que transforma les dades del processador (que es troben en paral·lel) a dades en sèrie, i viceversa. A part de transformar-les, afegeix els bits corresponents d'inici, fi, i totes les configuracions que ha de realitzar. Això resulta molt útil quan es tracta d'interactuar amb altres components que no hem programat nosaltres, i que per tant desconeixem quin format de dades utilitzen.

Les característiques d'aquest mòdul són força conegudes:

- Possibilitat de tenir rellotge intern o extern
- Independència entre el rellotge emissor i el receptor
- Emissió i recepció *full-duplex* en mode síncron o asíncron
- Receptor amb 4 *buffers*
- Emissor amb 2 *buffers*
- Format de dades programable:

  - Nombre de bits
  - Paritat
  - Bit de parada
- Possibilitat de treballar en diversos modes:
  - *Full-duplex*
  - *Automatic echo*
  - *Local loopback*
  - *Remote loopback*
- Activació automàtica al rebre dades
- Possibilitat de treballar amb interrupcions
- Accés a DMA
- Detecció d'errors
- Detecció i generació de canvis de línia

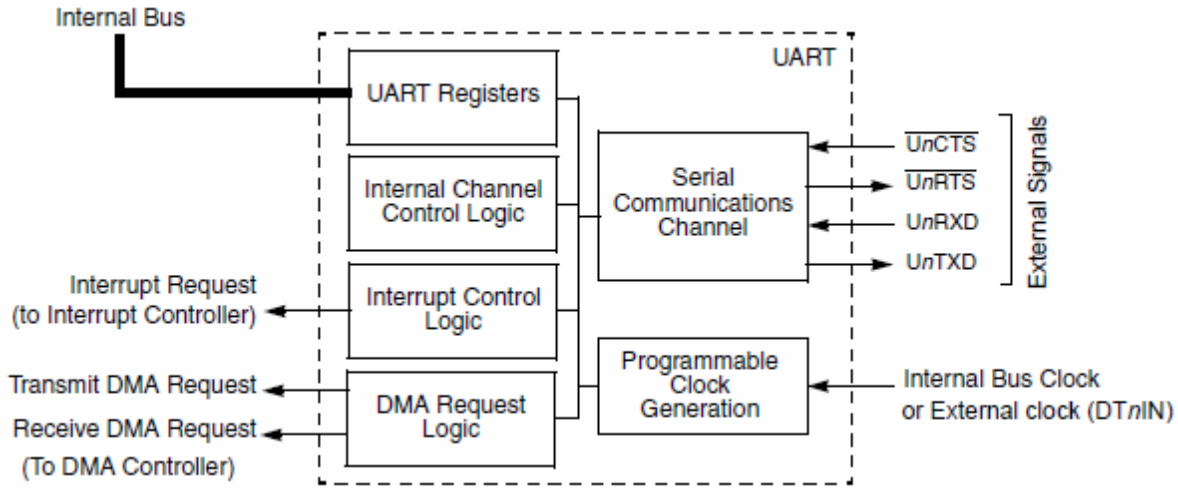

Figura 2.7: Diagrama de Blocs UART

Aquest mòdul és molt complet, com hem pogut comprovar, i el seu funcionament és força senzill dins del microcontrolador (figura 2.7), però utilitza un gran nombre de canals i és poc adequat per a un bus, ja que no disposa de gestor del bus ni control del canal.

2.3.3. I^2C

El bus I^2C ens ofereix la possibilitat d'interconnectar dispositius amb només 2 canals de dades. A més, permet la utilització simultània de més de 100 dispositius, molt més que suficient per a un sistema com el nostre.

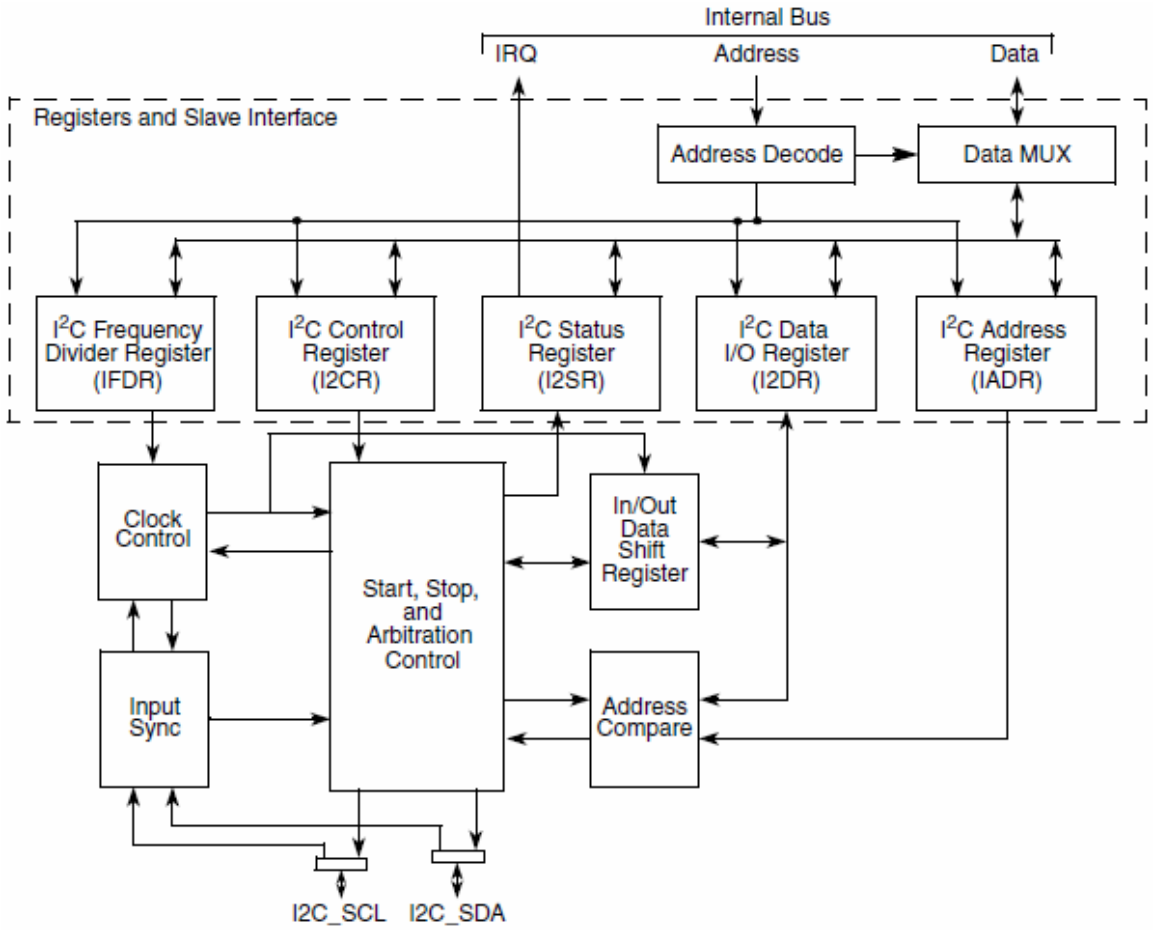

Figura 2.8: Diagrama de Blocs I2C

Les principals característiques d'aquest mòdul són:

- Compatibilitat amb l'estàndard 2.1 de I^2C
- Suport per a dispositius de 3'3v
- Funcionament amb més d'un *master*
- Rellotge intern programable
- Bit d'ACK
- Transferència *byte* a *byte*, controlada per interrupcions

- Canvi de *master* a esclau dinàmicament
- Interrupció d'ID rebut
- Generació de senyals d'inici i fi de transmissió
- Detecció de canal en ús

I^2C és molt popular per a la comunicació entre dispositius electrònics per que és molt senzill de configurar i el seu funcionament, a banda de poc costós en termes de computació, és força eficient. Amb només 2 canals és capaç de bastir un protocol que permet la comunicació entre un gran nombre de dispositius.

El seu funcionament dins del nostre microcontrolador és força senzill, i a través de pocs registres podem enviar correctament tota la informació desitjada.

2.3.4. FlexCAN

El mòdul FlexCAN ens permet la comunicació mitjançant el protocol Controller Area Network[1]. Inicialment aquest protocol es va dissenyar per a la comunicació dels dispositius integrats en un vehicle amb l'exterior, però s'ha anat ampliant fins a crear un protocol molt estable.

[1] CAN, en les seves sigles en anglès.

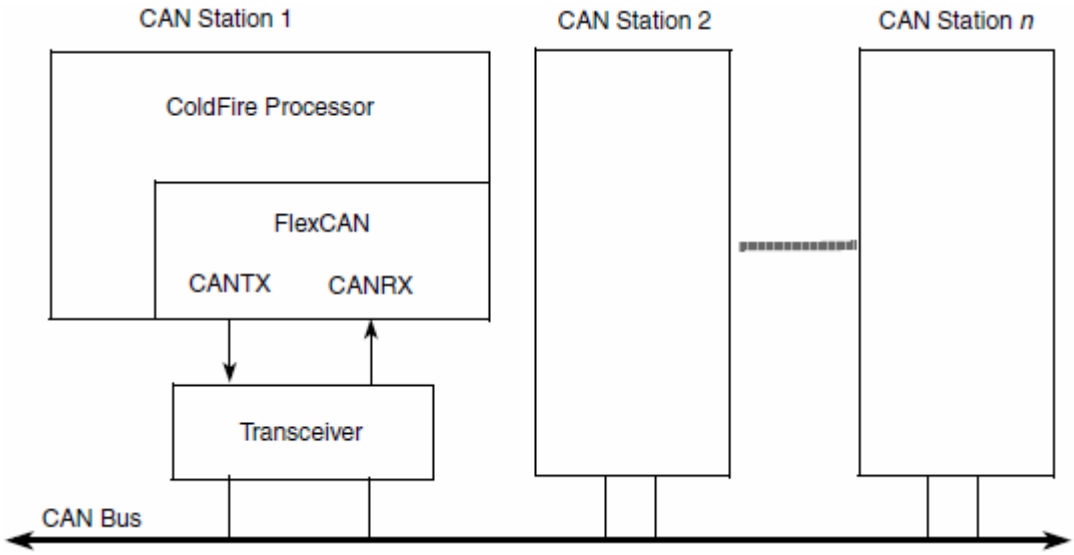

Figura 2.9: Bus FlexCAN

Bàsicament, cada node del sistema es troba connectat al CAN a través d'un adaptador que li transmet les dades (figura 2.9). En aquest sistema, tant els missatges com els nodes disposen d'identificadors únics.

Les característiques d'aquest mòdul són força avançades, ja que disposa d'una arquitectura de missatges molt potent:

- Compatible amb el protocol CAN v2.0B
    - Trames de 109 o 127 bits
    - Velocitat de 1Mbps
    - Adreçament relatiu al contingut
- 16 *buffers* de 8Bytes
- Mode de només lectura
- 3 registres programables per als *buffers*
- Dos esquemes de prioritats disponibles:
    - Segons ID
    - Segons número de *buffer*
- Marca de temps de 16 bits
- Sincronització del temps del sistema a través de missatges
- Modes I/O programables

- Gestor d'Interrupcions
- Independent del medi (sempre s'ha d'utilitzar un adaptador)
- Bus amb més d'un master
- Arquitectura de Xarxa lliure

El funcionament dins del microcontrolador es troba definit per l'esquema de la figura 2.10. Com podem veure, requereix un adaptador del medi i només disposa de dos canals de dades. L'accés al mòdul es fa a través de registres, i no hem d'entrar en detalls de *buffer* o d'estat del canal.

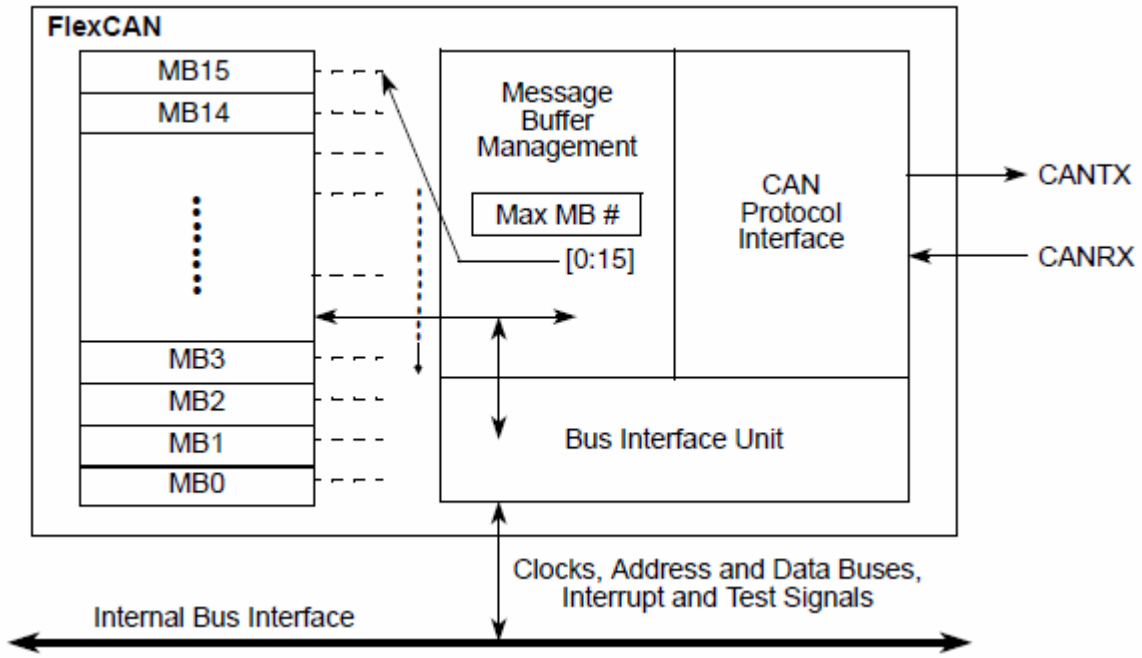

Figura 2.10: Diagrama de Blocs FlexCAN

Aquest bus és molt eficient, però tota la implementació dels missatges i les adreces queda una mica fora d'una aplicació com la nostra, que hauria de ser el més simple possible.

2.3.5. Tria del Bus de Dades

Un cop hem vist tots els possibles candidats i les seves característiques, per sobre, ha arribat el moment de prendre la decisió de quin bus implementar en el nostre sistema.

Si hem d'escollir el protocol més complet, sense dubte el CAN resulta el més adient, però ens carregarà massa el bus per només un tipus de dades que volem enviar.

Si el que volem és facilitats a l'hora d'enviar les dades, sigui quina sigui el seu format, el més adient és el port sèrie, és a dir, el mòdul UART. Aquest mòdul, però, no ens permet tenir més de dos nodes a la comunicació, per tant també queda descartat.

Les diferències entre QSPI i I^2C són d'arquitectura. Mentre que la primera utilitza 7 canals dels quals 4 són per a la selecció del node destí, la segona empra adreçament per evitar-se aquests bits de selecció. A més, mentre QSPI utilitza un canal d'entrada i un de sortida, I^2C converteix l'entrada i la sortida en un únic canal.
En quant a la implementació, I^2C resulta molt més senzill, ja que només es tracta d'indicar l'adreça i les dades, el microcontrolador s'encarrega de la resta. QSPI, en canvi, ha de seleccionar el node destí, activar-lo, assegurar-se que està actiu, etc.

Sembla clar, doncs, que per motius de senzillesa, baixa utilització de recursos, i gran capacitat d'adreçament emprarem el bus **I^2C** per a la comunicació del nostre sistema amb l'exterior.

3.Tractament del Senyal

El primer que cal fer quan ja hem decidit quins són els components que utilitzarem per a l'elaboració del sensor d'acceleració tridimensional és realitzar el tractament del senyal del sensor. L'objectiu d'aquest tractament és eliminar o reduir el possible error comès pel sensor.

Tots els sensors experimenten errors causats, entre d'altres, per la temperatura, la humitat o altres factors ambientals als que està sotmès. Com a conseqüència d'aquests fenòmens les dades que ens proporciona el sensor poden contenir errors, i per tant són susceptibles de ser corregides.

Estudiant el document d'especificacions del sensor d'acceleració escollit i sobretot la nota d'aplicació del mateix fabricant sobre la compensació de temperatura en acceleròmetres tèrmics hem establert un procediment per tal de reduir al mínim l'error comès pel sensor.

El procediment combina un procés de correcció, un procés d'ajust i un altre procés de filtrat del senyal. El primer ens adequa la senyal, el segon corregeix errors i l'últim ens elimina soroll de fons:

1. Correcció del coeficient de sensibilitat
2. Correcció de la desviació
3. Filtrat del Senyal

Bàsicament, quan obtinguem el valor d'acceleració que ens retorna el sensor li aplicarem, seqüencialment, la correcció del

coeficient de sensibilitat, la correcció de la desviació i finalment realitzarem un filtrat del senyal, tal i com mostra la figura 3.1.

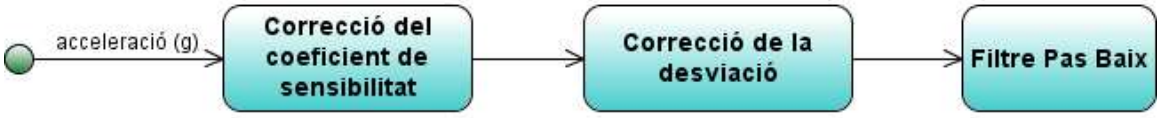

Figura 3.1: Diagrama de Tractament del Senyal

Les dues correccions són conseqüència de la temperatura, per això el sensor ja disposa d'un termòmetre incorporat que ens indica a quina temperatura es troba. Afortunadament, el comportament del sensor i, per tant dels seus errors, és sempre el mateix per a una temperatura donada, per exemple, a 50 graus centígrads sempre cometrà el mateix error, per tant es pot estudiar i corregir.

Per a obtenir la temperatura utilitzarem el mòdul Analògic-Digital (Analog-to-Digital Converter o ADC, en les seves sigles en anglès) del processador per tal de capturar el voltatge que surt de la pota de temperatura del sensor. En l'apartat de disseny de l'aplicació veurem amb quina freqüència i configuració elaborem la captura.
El procediment de calibració del termòmetre el trobem explicat amb detall en l'annex B d'aquest document.

En quant a l'obtenció de la senyal d'acceleració del sensor, emprarem una rutina que capturi l'ample dels polsos del tren de polsos que ens envia el sensor. Amb el temps de "on" del pols, podrem deduir l'acceleració aplicant aquests senzills càlculs:

$$A(t) = \frac{\frac{T_1}{T_2} - 0'5}{0'125} \qquad (3.1)$$

On, 0'125 és el coeficient de sensibilitat, T_1 és el temps de "on" i T_2 el període del pols, en el nostre cas, 10ms. Podem veure aquests elements representats en la figura 3.2.

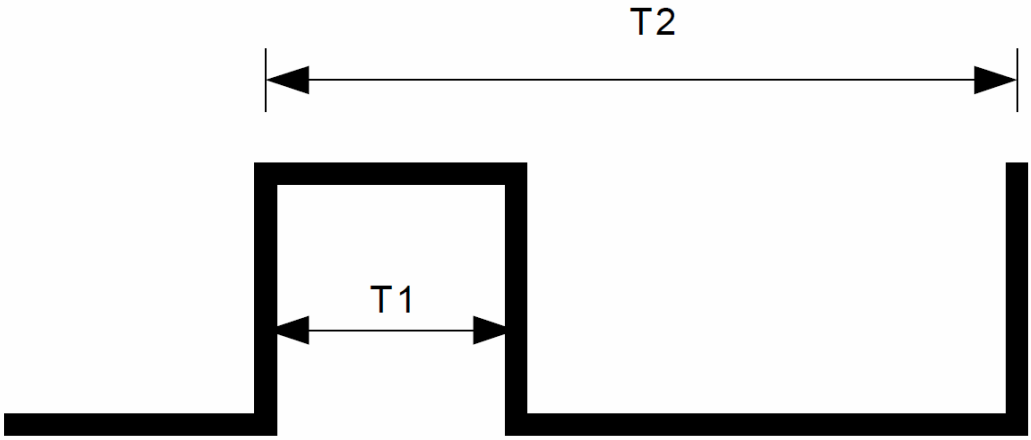

Figura 3.2: Període d'un Tren de Polsos

Coneixent ja els valors de les constants, podem operar amb l'equació per tal de trobar una forma més senzilla de càlcul tant per al nostre processador com per a l'hora de la simulació dels procediments de compensació de l'error que hem aplicat:

$$A(t) = t \cdot 800 - 4 \tag{3.2}$$

D'aquesta manera tindrem una senyal d'acceleració mostrejada a 100Hz, ja que el període del tren de polsos és de 10ms i cada pols ens indica el valor de l'acceleració.

Un cop ja hem definit l'origen de les dades, i el procediment bàsic per a obtenir-les, hem de fer una apreciació sobre el tractarem de les dades en aquest document: com hem vist, treballarem amb senyals mostrejades. Això és per que estem treballant amb components digitals i en un temps discret. Totes les aplicacions dels conceptes aquí exposats seran amb senyals discretes i mostrejades, però al llarg del document parlarem de "senyal" i no de mostres, per a facilitar la comprensió de les

explicacions. En tot moment, però es pot passar de senyals a mostres substituint la senyal per un seguit de mostres separades en el temps un període concret.

Entendrem, doncs, la senyal d'acceleració com una senyal continua del valor de l'acceleració que registra el sensor, i ens abstraurem que realment és un seguit de mostres codificades segons l'ample de pols d'un tren de polsos a una freqüència de 100Hz. També entendrem la senyal de temperatura com una senyal continua, no com els valors que ens dona el ADC del processador.

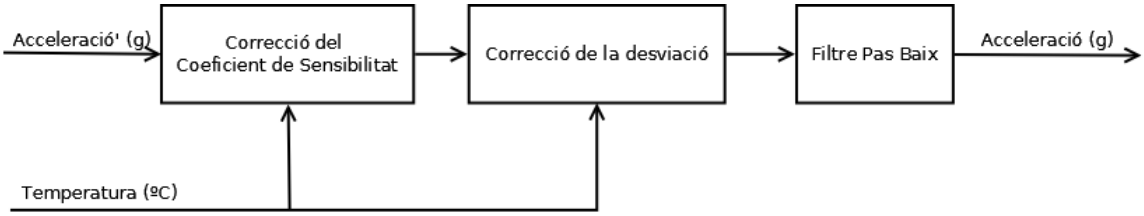

Figura 3.3: Diagrama de Senyals

Finalment, avaluarem els resultats obtinguts per cada un dels passos que estudiarem segons un joc de proves que hem definit. Enlloc de simular dades en MatLab, hem considerat més adequat treballar amb dades reals, extretes directament del sensor, tant acceleracions com temperatures.

Aquests jocs de proves consisteixen en dues situacions extremes a les que hem considerat que el sensor havia de mostrar una resposta raonable.
No creiem que el sensor s'hagi de trobar en cap de les dues situacions al llarg de la seva vida útil, degut a la intensitat del canvi de temperatures: en el primer cas, en que decidim escalfar el dispositiu, passarà de 20'26ºC (1'22V) a 40'26ºC (1'32V), és a dir, un augment de 20º centígrads en 5 segons (figura 3.4).

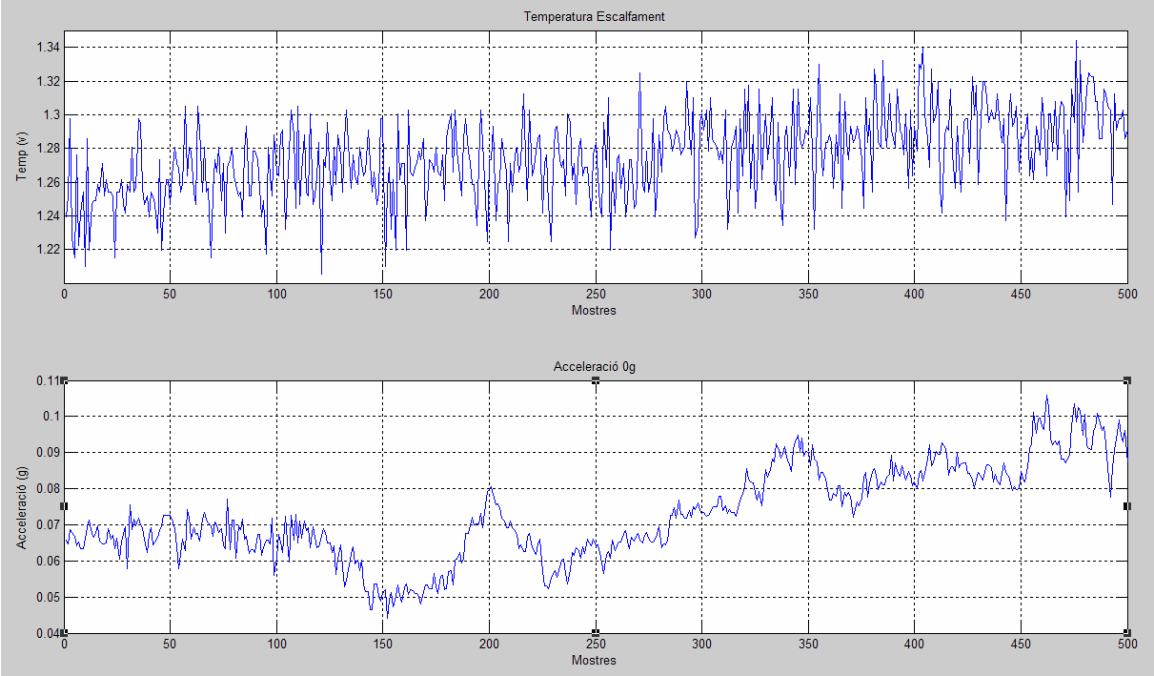

Figura 3.4: Prova 1. Augment de la Temperatura

En el segon cas refredarem el sensor també d'una forma brusca: passarem de 16'26ºC (1'2V) a 2.26ºC (1'13V). En aquest cas, fins i tot es baixa més d'aquest punt, com podem comprovar en la figura 3.5 sobre la mostra número 100, ocasionant alhora una gran desviació en l'acceleració.

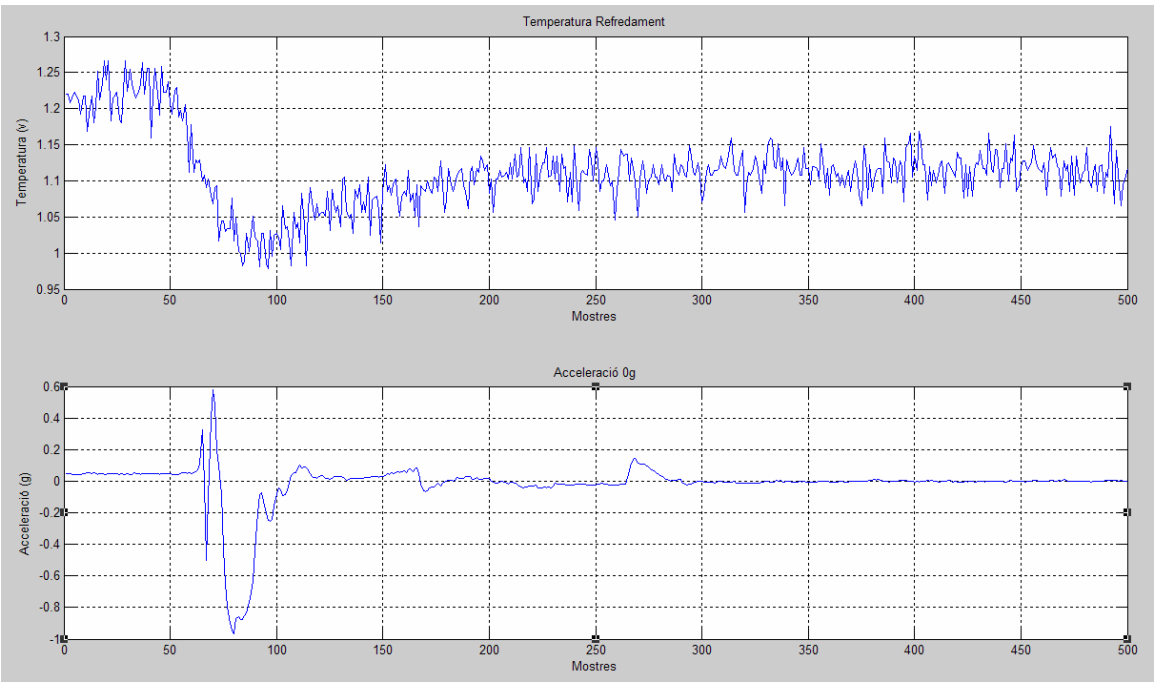

Figura 3.5: Prova2. Disminució de la Temperatura

D'aquesta manera, hem fet treballar el sensor en pràcticament totes les temperatures del seu rang de treball i hem capturat tant la temperatura com la senyal d'acceleració obtinguda amb el sensor en repòs.

Altre cop, tractarem tant la temperatura com l'acceleració com senyals contínues, de la mateixa forma que farem en la resta del document. En els següents apartats analitzarem en detall les característiques dels tres procediments aplicats per a la correcció del senyal.

3.1. Correcció del Coeficient de Sensibilitat

La sensibilitat d'un equip de mesura és la variació més petita que l'equip pot registrar. En el cas del nostre sensor es tracta de l'acceleració més petita que pot representar.

3.1.1. Fonaments Teòrics

Com en la majoria de tecnologies disponibles, els acceleròmetres tèrmics varien la seva sensibilitat, segons la temperatura a la que es troben, al variar el seu coeficient de sensibilitat. Afortunadament, aquesta variació és previsible i repetible, per tant, podem calcular-la i corregir el coeficient adequadament. De fet, aquesta variació no depèn de les característiques pròpies de cada xip, sinó que és la mateixa per tots els sensors que s'han fabricat seguint el mateix patró.

Per a facilitar la obtenció de dades d'acceleració sense haver de realitzar compensacions amb la temperatura, el fabricant ens ha donat una equació on la temperatura no hi apareix, però sí un coeficient de sensibilitat (0'125). Aquest coeficient és el coeficient de sensibilitat a 25ºC, però si el sensor es troba a una altra temperatura, estarem cometent un error, ja que el coeficient hauria de ser un altre.

A grans trets, podríem dir que a major temperatura el coeficient de sensibilitat ha de ser menor, mentre que si baixem la temperatura del xip, haurem d'augmentar-lo. Més concretament, segueix la següent equació:

$$S_i \cdot T_i^{2'81} = S_f \cdot T_f^{2'81} \qquad (3.3)$$

On S_i és el coeficient de sensibilitat a una temperatura T_i i S_f és el coeficient de sensibilitat a una temperatura T_f, sempre en graus Kelvin. L'exponent 2'81 és característic de cada sèrie de sensors, per exemple, en el cas dels sensors amb reducció de soroll és 2'81 mentre que per als sensors que no tenen aquesta reducció és 2'67.

Podem representar el coeficient de sensibilitat normalitzat, és a dir, el valor del coeficient relatiu al coeficient de sensibilitat a 25 graus centígrads (que és 0'125). Considerant que el coeficient a aquesta temperatura és 1, la gràfica de coeficients resultant és la que podem apreciar a la figura 3.6.

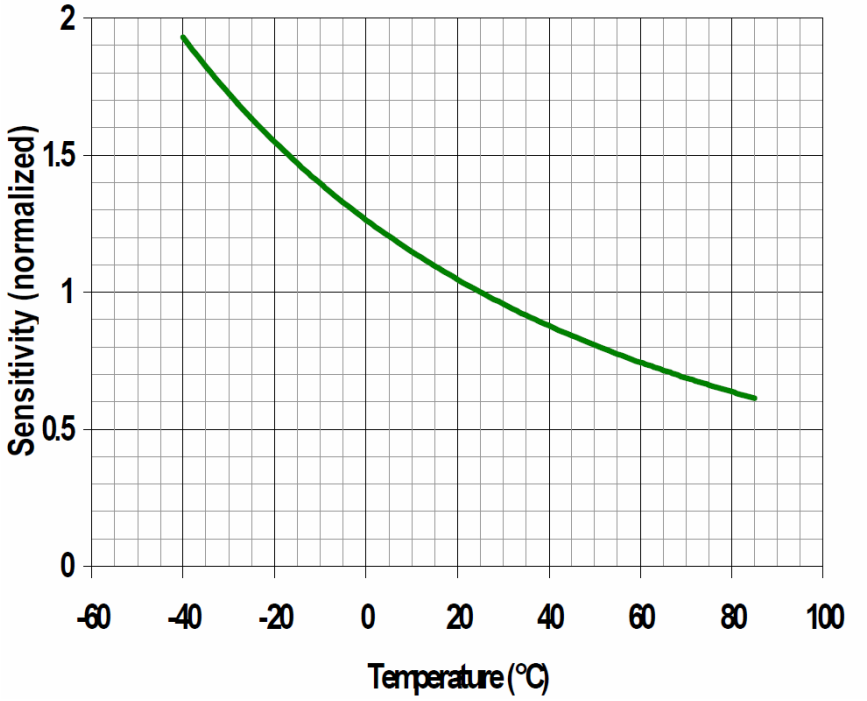

Figura 3.6: Coeficient de Sensibilitat Normalitzat

Si entenem com a sensibilitat el coeficient de sensibilitat, podem dir que la sensibilitat a 25 graus és 1, mentre que si augmentem la temperatura, la sensibilitat disminueix i si disminuïm la temperatura, la sensibilitat augmenta.

En el cas del nostre sensor, el rang de treball és de 0º C a 70º C, per tant la sensibilitat oscil·la entre 1'25 i 0'7, respectivament.

Com podem veure en l'equació de l'obtenció de l'acceleració, sempre s'utilitza el coeficient de sensibilitat propi dels 25ºC. El que hem de fer, doncs, és adaptar aquest coeficient al coeficient corresponent a la temperatura a la que es troba el sensor.

El fet de que el coeficient de sensibilitat normalitzada sigui 1 per a una temperatura de 25º centígrads és exactament per que aquest és el coeficient de sensibilitat que s'utilitza per a fer la transformació de temps a acceleració en cas de no tenir en compte la temperatura.

Al corregir aquest coeficient, en definitiva, el que farà és "potenciar" les acceleracions capturades en condicions tèrmiques que coneixem favorables per al bon funcionament del sensor, mentre que disminuïm el pes d'aquelles captures amb una major probabilitat de ser errònies, o si més no d'estar desviades.

3.1.2. Elaboració d'una Rutina d'Aplicació

Un cop hem entès els conceptes relacionats amb la sensibilitat i el coeficient de sensibilitat, podem passar a elaborar una petita rutina que ens permeti corregir aquest coeficient. Basats en l'equació del coeficient de sensibilitat, no és difícil deduir aquesta altra:

$$S_f = S_i \cdot \frac{T_i^{2'81}}{T_f^{2'81}} \qquad (3.4)$$

Ara hem de trobar per a una temperatura determinada, un coeficient concret i utilitzar-lo com a referència. Per a facilitar els

càlculs, emprarem el fet que a 25º centígrads el coeficient val 1, així doncs,

$$S_f = 1 \cdot \frac{T_{25}^{2'81}}{T_f^{2'81}} \tag{3.5}$$

Finalment, per corregir el coeficient de sensibilitat, només hem de multiplicar-lo per aquest S_f resultant. Hem de fixar-nos ara, que en cas de tenir com a T_f, és a dir, com a temperatura actual, 25ºC, el coeficient de sensibilitat S_f valdria exactament 1, per tant, no caldria modificar el coeficient utilitzat per defecte, ja que per defecte sempre utilitzem un coeficient de sensibilitat de 25ºC, 0'125.

Si incorporem ara a magnitud S_f a l'equació original del càlcul de l'acceleració,

$$A_{compensada}(t) = \frac{\frac{T_1}{T_2} - 0'5}{S_f \cdot 0'125} \tag{3.6}$$

O el que és el mateix, desenvolupant l'equació de S_f:

$$A_{compensada}(t) = \frac{\frac{T_1}{T_2} - 0'5}{\frac{T_{25}^{2'81}}{T_f^{2'81}} \cdot 0'125} = \frac{\frac{T_1}{T_2} - 0'5}{0'125} \cdot \frac{T_f^{2'81}}{T_{25}^{2'81}} \tag{3.7}$$

Finalment, podem simplificar l'equació de tal forma que, per a corregir el coeficient de sensibilitat, el que hem de fer és:

$$A_{compensada} = A \cdot \frac{T_f^{2'81}}{(25 + 273'15)^{2'81}} \tag{3.8}$$

Com podem observar, es tracta d'una conversió exponencial, és a dir, per obtenir l'acceleració compensada hem de calcular expressions exponencials. Evidentment, el denominador de la

fracció és una constant, i no caldrà calcular-la per cada acceleració, però el denominador caldrà calcular-lo en cada instant.

Existeix la possibilitat d'aproximar aquesta expressió a una equació d'una paràbola, de tal forma que sigui una expressió polinomial, per tal de reduir el temps de còmput i per tant la càrrega del processador. Una de les avantatges de treballar amb xips de les característiques de l'escollit és que no cal fer aproximacions que ens facin perdre resolució en un compromís amb el rendiment. A un baix cost disposem de processadors suficientment potents per a realitzar els càlculs originals.

En resum, el que haurem de fer per tal de corregir el coeficient de sensibilitat és aplicar la següent fórmula a totes les acceleracions obtingudes del sensor:

$$A_{compensada} = A \cdot \frac{T_f^{2'81}}{8'9776 \cdot 10^6} \qquad (3.9)$$

3.1.3. Proves Realitzades

Un cop ja hem elaborat la rutina, podem utilitzar les senyals de prova que hem capturat per tal d'avaluar el comportament d'aquesta correcció. Es d'esperar que no ens proporcioni un ajust a zero perfecte, però que si, en teoria, ens hauria de suavitzar les formes rebaixant-les cap a zero.

En el cas de l'escalfament del sensor (figura 3.7), podem veure com la senyal de l'acceleració no acaba de ser mai zero (sempre hi ha offset) però després de la correcció de la sensibilitat aquesta acceleració és més propera a zero. A més, sembla que

redueix també l'amplitud de la cresta de la senyal en les zones on hauria de ser més contínua.

Si ens fixem ara en el cas del procés en que refredem el sensor (figura 3.8) veurem com, en la major part de la captura, la senyal original és força propera a zero.

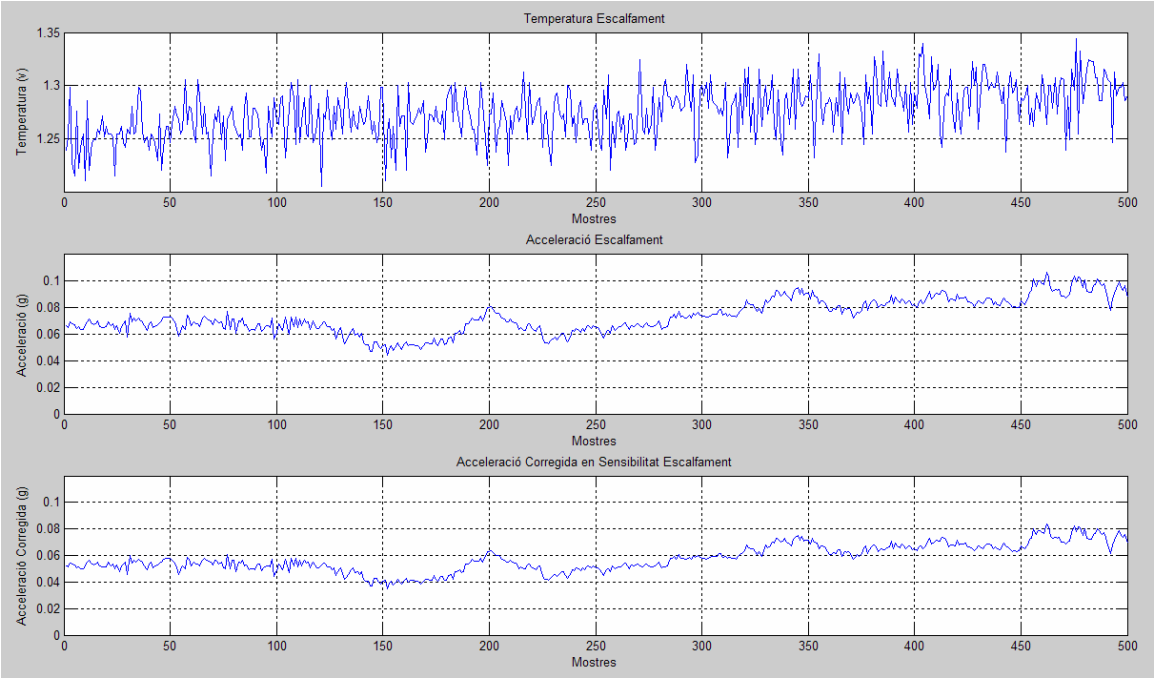

Figura 3.7: Sensibilitat. Prova 1 Escalfament

S'ha d'exceptuar, però, la franja en la que es realitza una baixada molt gran de temperatura. Si ens fixem en la correlació que hi ha entre la senyal de temperatura i la de l'acceleració, veurem que entre les mostres 50 i 100 es produeix un descens molt pronunciat de la temperatura. En volts és, amb prou feines, 0'1 volts, però tenint en compte que cada 5mV és un grau, estem parlant d'una variació de 20º centígrads. Aquest canvi tant sobtat fa que el sensor no enregistri correctament el valor de 0g, però en canvi, en quan s'estabilitza, ens retorna un valor molt pròxim a zero.

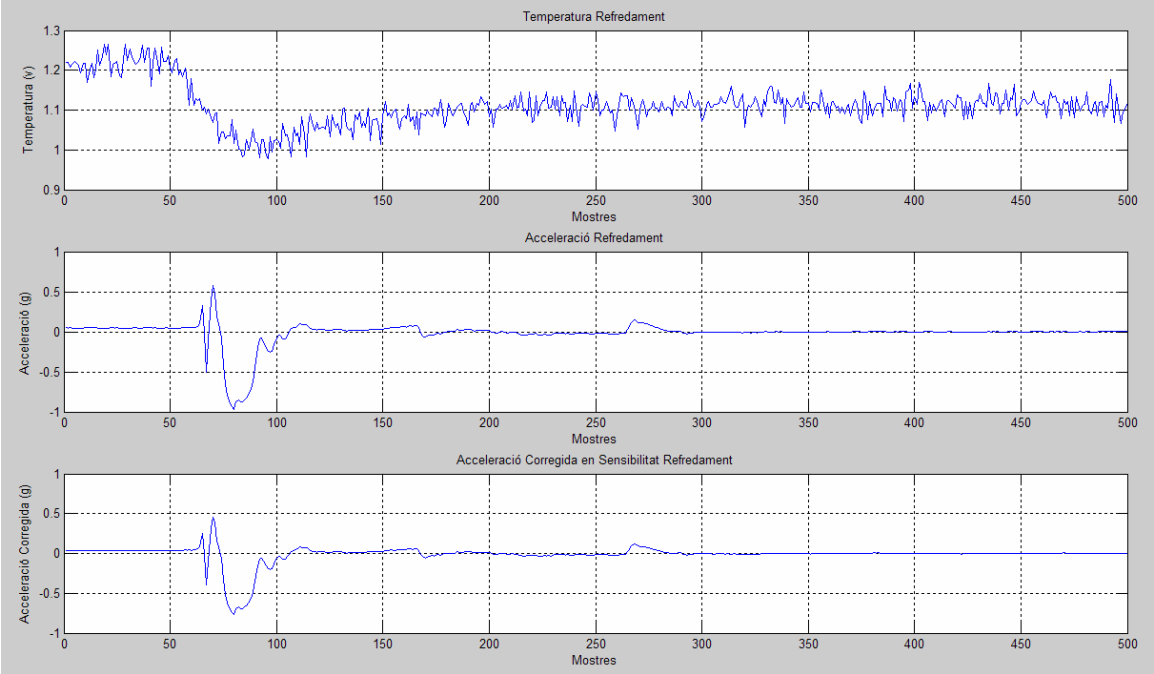

Figura 3.8: Sensibilitat. Prova 2 Refredament

Finalment, l'aplicació de la correcció de sensibilitat millora la senyal en totes les situacions, encara que no elimina l'offset que tenim present. Analitzarem aquest soroll en el següent apartat.

3.2. Correcció de la Desviació

Aquesta segona part del procés de tractament del senyal intenta corregir les desviacions que tenen els sensors per problemes de temperatura. S'anomena, també, correcció de la desviació a 0g per que s'utilitza la senyal que llegim del sensor quan no hi ha acceleració, és a dir, quan el sensor es troba en repòs. Anomenarem *offset* a aquesta senyal diferent de zero que trobem quan el sensor està aturat.

Tot i que només analitzarem l'acceleració registrada en repòs, les desviacions del sensor es produeixen al llarg de tot el domini de treball d'aquest i, per tant, la correcció de la desviació s'aplica en tot el rang de valors possibles d'acceleració, des del repòs fins a l'acceleració màxima que pot registrar el sensor.

Les desviacions sofertes per un sensor són característiques pròpies d'aquest sensor, és a dir, caldrà ajustar els valors de la correcció per cada unitat diferent que utilitzem.

Les desviacions produïdes per les variacions de temperatura es poden obviar si l'aplicació que estem desenvolupant no requereix mantenir un històric de dades. Si per contra, caldrà enregistrar totes les acceleracions per tal de realitzar algun càlcul amb un nombre gran de mostres, aquesta variació ens pot alterar completament el valor final.
Suposem, per exemple, que capturem l'acceleració en cada instant i que després volem integrar totes aquestes captures per tal d'obtenir la velocitat actual. En el cas d'estar tota l'estona parats, no hauria d'aparèixer cap acceleració, i per tant la velocitat hauria de ser zero. Si tenim un *offset*, per petit que sigui, la integral d'una

constant al llarg de suficient temps acabarà sent infinit, i el sensor no s'haurà mogut.

Tenint en compte que les desviacions depenen de la temperatura (T) i que són predictibles i reproduïbles, podem definir una funció, Z(T), que ens indiqui la desviació que sofrirà el sensor a la temperatura T.
Coneixent el valor de la desviació per qualsevol temperatura, podrem, per cada mostra capturada i segons a la seva temperatura, restar-li el valor adequat i obtenir l'acceleració lliure d'error:

$$A_{compensada} = A - Z(T) \qquad (3.10)$$

El que farem en els següents apartats és estudiar les desviacions i trobar la forma en la que es relacionen amb la temperatura.

És important que no oblidem en quin punt del procés del tractament del senyal ens trobem. Si revisem l'esquema de funcionament (figura 3.3) veurem que estem just després de la correcció del coeficient de sensibilitat, així doncs, cal que recordem que totes les dades d'acceleració que tractem aquí ja han estat corregides en sensibilitat.

3.2.1. Fonaments Teòrics

Consultant la nota d'aplicació del fabricant trobem una valoració de la magnitud d'aquestes desviacions de l'acceleració. Concretament, i per motius obvis de facilitat de captura de dades, ens mostra gràficament el rang de desviacions que pot prendre a 0g, és a dir, en repòs. La zona ombrejada de la figura 3.9 són els

possibles valors que pot prendre l'acceleració mentre es troba el sensor en repòs, segons les temperatures indicades en l'eix d'ordenades.

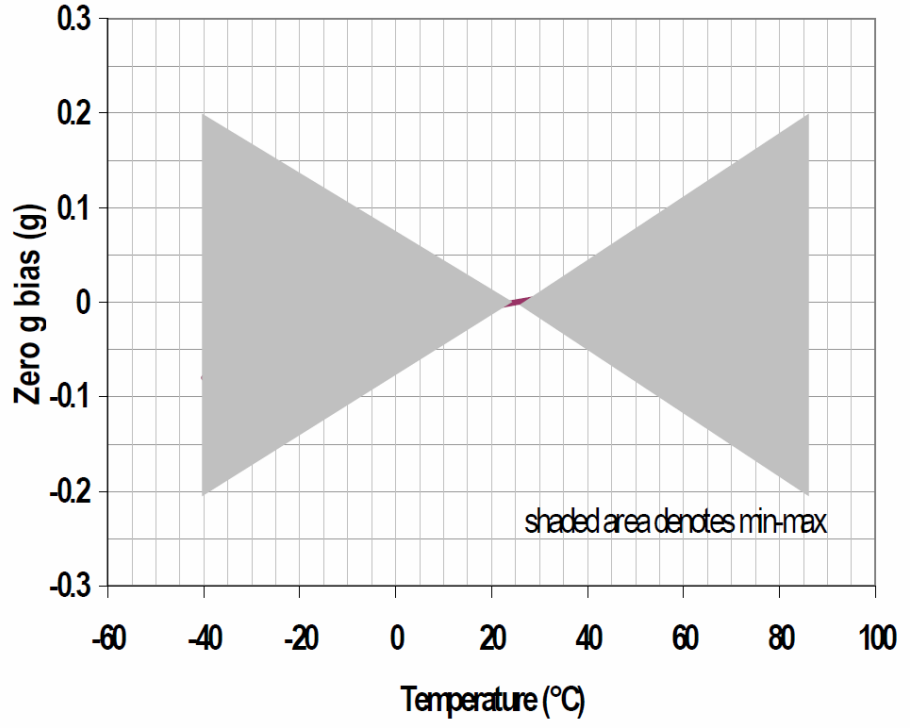

Figura 3.9: Desviació a 0g

Fixem-nos que aquest cop són les temperatures extremes les que ocasionen majors desviacions sobre el valor real, independentment del seu signe.

Mentre en la sensibilitat com més fred estava el sensor major havia de ser el coeficient de sensibilitat per a proporcionar el valor correcte, ara, els valors òptims els trobem a temperatures que ronden els 25ºC, mentre que com més ens allunyem d'aquestes temperatures, major serà la desviació enregistrada.

Aquest offset, tot i que pot semblar petit, ens pot arribar a indicar ±0'2g en repòs, això és, aproximadament, ±1'96m/s^2. Tot i que no representa una gran acceleració, és un valor força provable per a la nostra aplicació, per tant no ens podem permetre tenir aquest valor per "error".

La nota d'aplicació del fabricant afirma que podem caracteritzar les desviacions del sensor mitjançant una paràbola, és a dir, una expressió de segon grau, on tenim dos coeficients i un terme independent per determinar:

$$Z(T) = a + b \cdot T + c \cdot T^2 \tag{3.11}$$

Sabent que les desviacions són predictibles i repetibles, un cop haguem solucionat les tres incògnites característiques de cada unitat, podrem aproximar l'error comès per a qualsevol temperatura.

El mètode per solucionar les tres incògnites és obvi: trobar tres valors per als que coneguem la temperatura i la desviació. Per a fer-ho, haurem de portar el sensor a tres temperatures diferents i mesurar l'offset en cada cas. Amb les tres mostres, podrem trobar les tres incògnites.

Amb els valors en els tres estats, podem trobar els valors de les incògnites a mà, però preferim fer-ho amb l'eina MatLab, que ens ofereix la representació de la corba i els valors d'ajust.

3.2.2. Caracterització de la Paràbola

Haurem de portar, doncs, el sensor a tres temperatures diferents. El més senzill és aprofitar la temperatura ambient per a realitzar una captura, aleshores refredar el dispositiu i prendre una altra mostra, i finalment escalfar-lo a una temperatura superior a la ambient i realitzar la última captura.

Fer una captura a temperatura ambient no és gens difícil, només cal mesurar l'acceleració i la temperatura que ens retorna el

xip, ens centrarem en el procediment de captura de les altres dues situacions.

La major dificultat a l'hora d'alterar la temperatura del sensor no és modificar-la, sinó mantenir-la el temps suficient per a que el sensor s'estabilitzi i ens retorni, només, el valor de l'offset degut a la temperatura a la que es troba, i no a la ràpida transició que ha realitzat.

Per a veure un exemple més clar, podem remitir-nos als casos de prova que hem exposat en la introducció d'aquest capítol. Concretament, en el moment de refredar el dispositiu, hem vist com quan hi ha hagut un descens sobtat de la temperatura, la senyal d'acceleració s'ha vist alterada igualment, però que al cap d'unes 30 mostres, s'ha estabilitzat la temperatura i la senyal s'ha tornat a aproximar a zero.

Si el que volem és capturar només l'*offset* produït per la temperatura a la que es troba el dispositiu i no per el canvi de temperatura realitzat, hem de deixar que la temperatura s'estabilitzi.

De la mateixa forma, si volem que el sensor s'estabilitzi, li hem de proporcionar un ambient igualment estable a la temperatura a la que volem que s'estabilitzi. Aconseguir un ambient tancat, a una temperatura homogènia i estable, encara que només sigui per un parell de segons, representa un repte que no és trivial.

Refredar el Sensor

En el cas de refredar el sensor, emprarem el principi físic de que un gas en expansió absorbeix calor (energia) de l'ambient. En el nostre cas utilitzarem aire comprimit. Si apliquem directament un flux d'aire comprimit al sensor, aconseguirem que la seva

temperatura disminueixi, però no podrem estabilitzar-la per sobre dels zero graus.
Si mantenim el flux constant, la temperatura s'estabilitzarà, però serà sota zero, ja que mentre l'expansió de l'aire pugui continuar absorbint calor, la temperatura del xip continuarà baixant, i no serà fins a estar diversos graus sota zero que deixarà de poder absorbir calor.
Com que el rang de treball del nostre sensor és per a valors superiors als zero graus centígrads, aquesta opció no és viable. Es fa indispensable construir una petita cambra per a mantenir la temperatura del sensor constant durant un parell de segons.

Per a la construcció d'aquesta petita cambra s'utilitza un bloc de poliestiré expandit al que se li ha realitzat un petit forat a la base, de la mida aproximada del sensor, i amb un altura suficient per a encabir-lo-hi. A més, s'ha preparat una petita conducció des del forat fins a l'altra cara del bloc per tal de poder passar el tub per on introduirem l'aire comprimit. Així doncs, el petit forat que hem fet farà de cambra amb temperatura controlada i allí és on es trobarà el nostre sensor.

Un cop preparat el bloc, el situarem sobre el sensor i introduirem aire comprimit per l'obertura. Veurem com la temperatura baixa, quan haguem arribat a un valor pròxim, però superior, a zero, deixarem d'introduir aire i veurem com la temperatura del sensor s'estabilitza durant uns segons. En aquest moment haurem de capturar l'acceleració.

Escalfar el Sensor

Si per a refredar el sensor disposem d'eines com l'aire comprimit, per escalfar-lo només podem aplicar aire calent.

Desgraciadament no és tant senzill canalitzar l'aire calent com l'aire comprimit, per tant, no podrem utilitzar la mateixa cambra de temperatura controlada que hem dissenyat per a l'altre cas.

Per escalfar el sensor utilitzarem un flux d'aire calent. El procediment serà el mateix: escalfar el sensor, esperar a que s'estabilitzi i prendre la mostra. Per escalfar el sensor li aplicarem aire calent, però el problema el trobarem a la fase d'estabilització.
Al no disposar de cap cambra, haurem de simular un ambient a temperatura controlada. El que farem és basar-nos en el fet que el sensor estarà a una temperatura lleugerament inferior a la de l'aire calent que li estem aplicant. Si apliquéssim aquest aire durant suficient temps (això és, molt temps) el sensor acabaria estant a la mateixa temperatura que l'aire, però fins arribar a aquest punt, el sensor sempre estarà una mica més fred que l'aire aplicat.
Això ho aprofitarem per fer assolir una temperatura determinada al sensor i immediatament baixar la temperatura de l'aire aplicat a aproximadament la temperatura a la que es trobi realment el xip. D'aquesta manera, el sensor no alterarà la seva temperatura, si més no, durant un parell de segons. Aquest serà el moment en que nosaltres prendrem la mostra.

Obtenció dels Coeficients de la Corba

Un cop tenim les dades a les tres temperatures, emprarem el programa MatLab per tal de realitzar l'ajust quadràtic. El propi programa ens donarà els valors dels coeficients a, b i c.

Dins del programa MatLab trobem l'aplicació *Curve Fitting Tool* (*cftool*) que és la responsable de realitzar ajustos de funcions. Bàsicament ens ofereix una pantalla per a la introducció de dades i una altra per a definir l'ajust i mostrar els resultats (figura 3.10).

Introduïm, doncs, la temperatura dels tres casos com a valor de l'eix d'ordenades i l'acceleració (un cop corregida en sensibilitat) com a valor de l'eix d'abscisses. A la finestra d'ajust, indiquem que s'ha de realitzar un ajust quadratic polynomial (també anomenat de LaGrange).

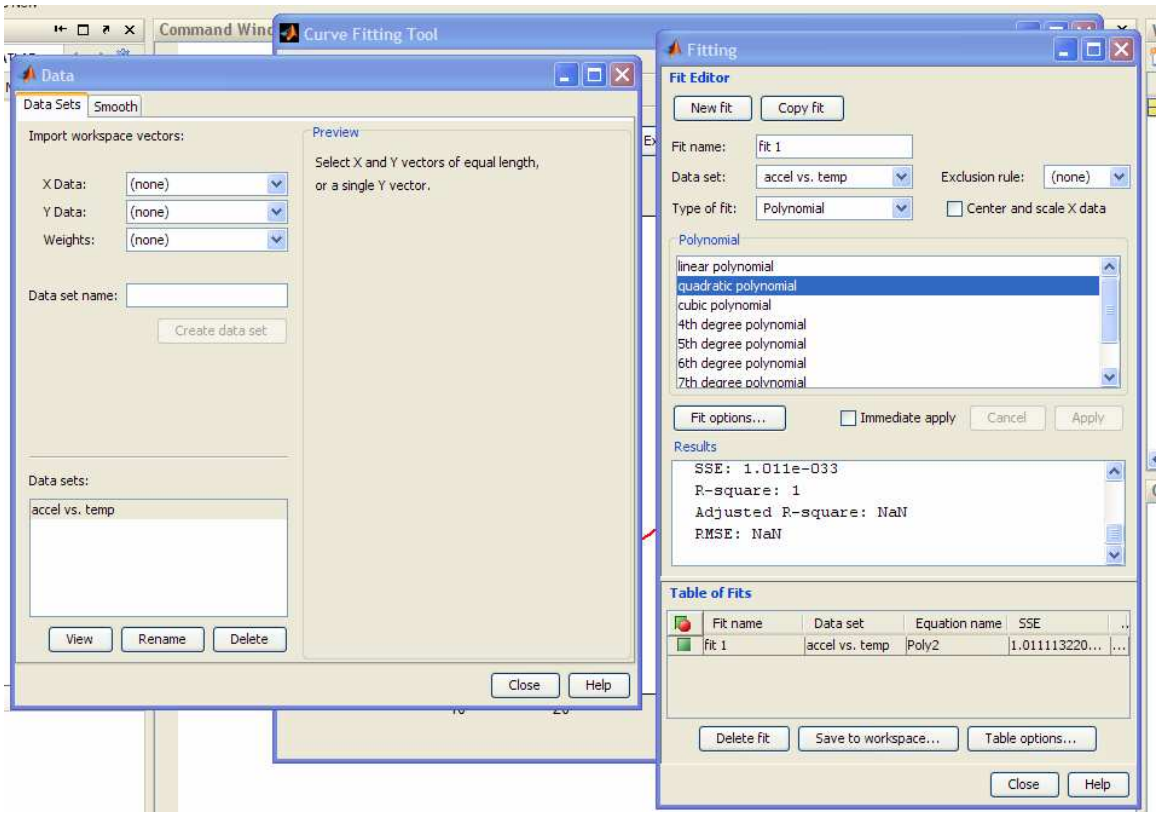

Figura 3.10: Eina Curve Fitting de MatLab

Si apliquem els canvis, la pròpia aplicació ens donarà els valors dels coeficients i els valors de l'ajust de la corba. Com es tracta d'una corba definida per tres punts, l'índex de regressió és 1, ja que la corba passa exactament pels tres punts.
També ens mostra una representació de la corba (figura 3.11), on podem veure la seva evolució segons les temperatures.

Com hem vist, només podem assegurar que la desviació és exactament la indicada per la corba en els punts mesurats, però si coneixem que la forma de la desviació és la d'una paràbola, podem

extrapolar quina serà la desviació en els altres punts sense haver de mesurar-la.

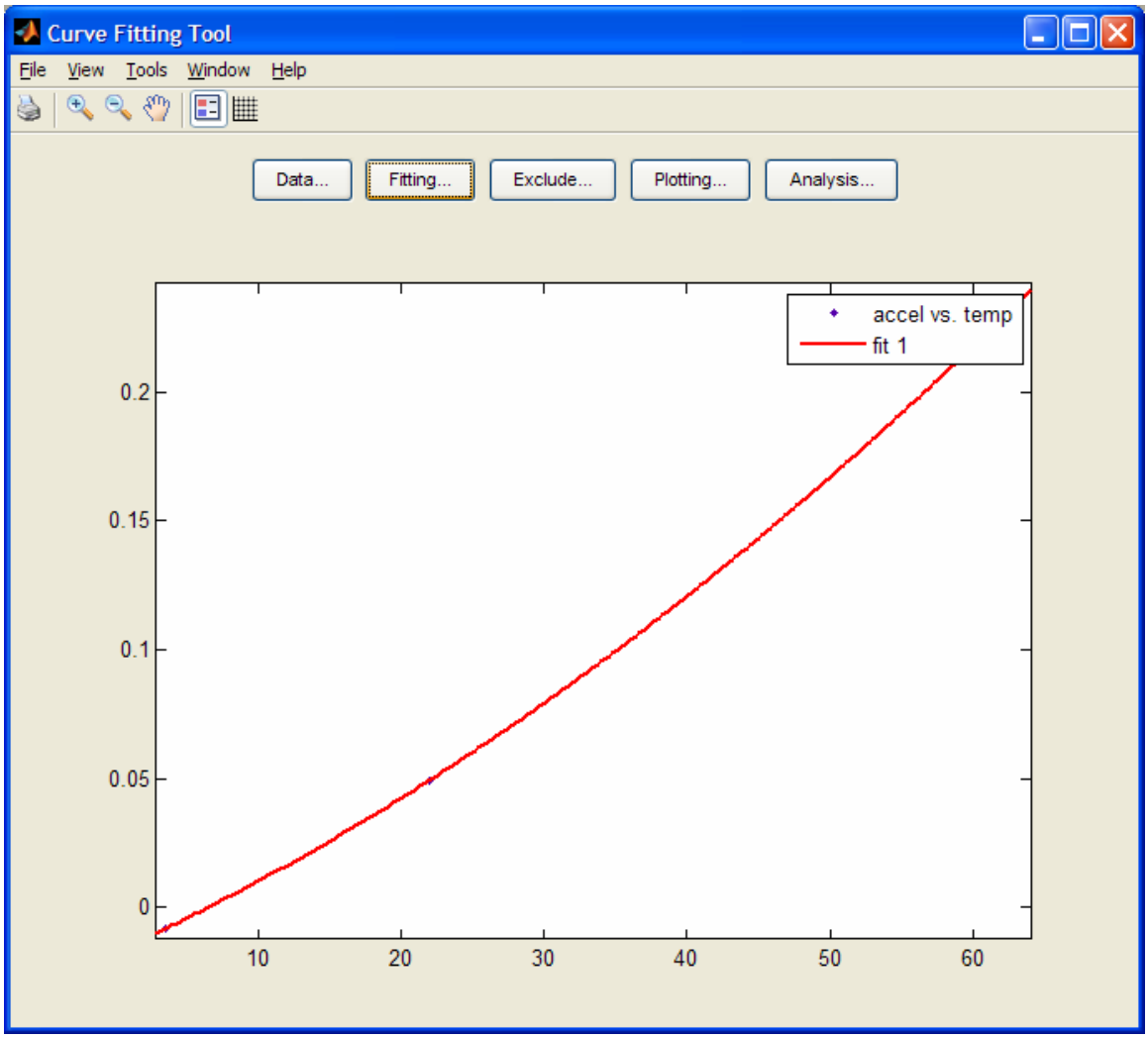

Figura 3.11: Ajust de la Corba

Un altre procediment per a conèixer la desviació en tots els punts podria consistir en l'elaboració d'una taula amb les diferents mesures i consultar-la cada vegada, però això implicaria realitzar moltes mesures per cada sensor, amb la conseqüent dificultat que això comporta. A més a més, al tractar-se de valors continus, sempre hauríem de decidir amb quina precisió volem treballar i elaborar extrapolacions quadràtiques per als casos intermitjos que no arribem a mesurar.

Un cop calculats els coeficients hem de definir una rutina practicable pel processador per aplicar aquesta correcció.

3.2.3. Elaboració d'una Rutina d'Aplicació

Ara que ja tenim l'expressió de la desviació definida i caracteritzada per al nostre sensor, ens queda definir una rutina per tal de que el processador pugui aplicar aquesta correcció a cada mostra. Comencem per recordar què hem de fer amb la funció Z(T):

$$A_{compensada} = A - Z(T) \tag{3.12}$$

On, A és l'acceleració que ens retorna el sensor i corregida en el coeficient de sensibilitat, també segons la temperatura,

$$Z(T) = a + b \cdot T + c \cdot T^2 \tag{3.13}$$

Finalment, per a cada acceleració que capturem del sensor, caldrà aplicar el següent factor de correcció:

$$A_{compensada} = A - (a + b \cdot T + c \cdot T^2) \tag{3.14}$$

Cal recordar que en tots els càlculs realitzats en aquest apartat es suposa que ja s'ha realitzat la correcció de sensibilitat, i que l'ordre de les correccions no es pot alterar, ja que els factors a, b i c estan calibrats segons l'acceleració ja corregida en sensibilitat.

Per avaluar el polinomi Z(T) podríem aplicar alguna funció d'avaluació que ens reduís el temps de còmput, però al tractar-se només d'haver d'elevar T al quadrat i operar-la amb els coeficients, la crida a la funció i tot el sobrecòmput que això comporta, fa que el temps de còmput total sigui el mateix.

Dissenyada ja la rutina que ens compensa les desviacions segons la temperatura, ha arribat el moment de comprovar el correcte funcionament d'aquesta correcció amb els jocs de proves definits.

3.2.4. Proves Realitzades

Com en el cas de la correcció del coeficient de sensibilitat, hem d'avaluar si aquesta correcció és efectiva o bé ens empitjora la qualitat del senyal. De la mateixa forma que en el cas anterior, aplicarem la correcció als nostres jocs de proves i compararem el resultat original amb el resultat d'aplicar la nostra correcció.

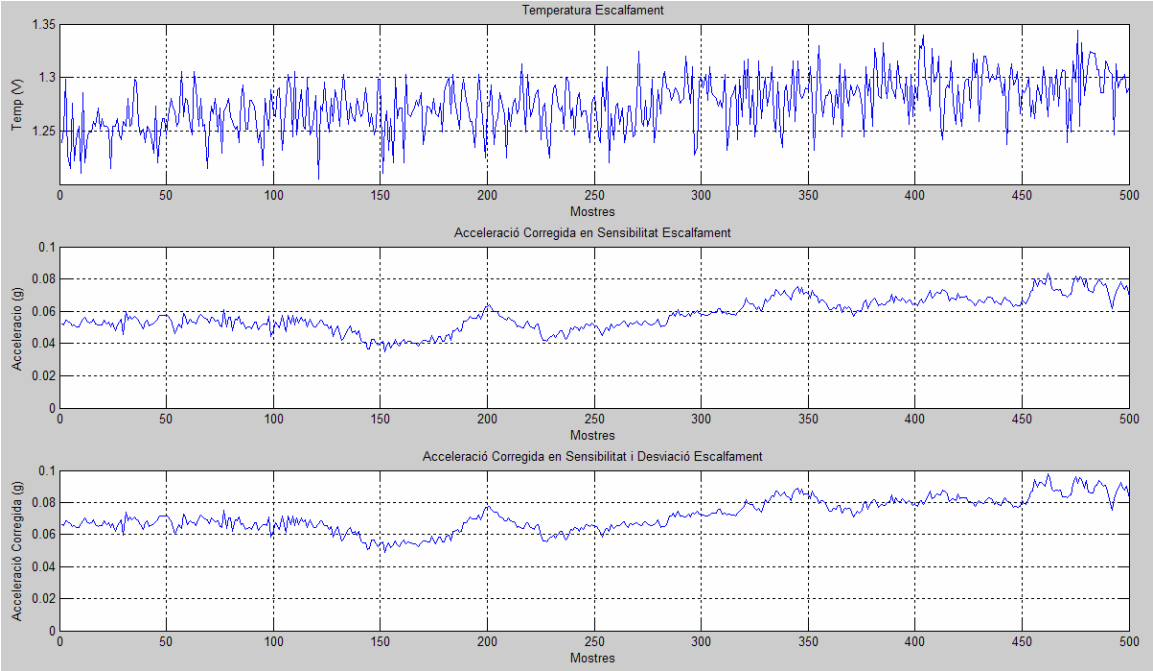

Figura 3.12: Desviació. Prova 1 Escalfament

En el cas de l'escalfament, podem observar com la correcció del coeficient de sensibilitat és molt més eficaç que la correcció de les desviacions. De fet, en aquest cas, en el que estem estudiant un comportament extremadament dinàmic de la temperatura del

sensor, el resultat d'aplicar aquesta correcció és contraproduent, tal i com podem veure en la figura 3.12.

Com ja hem indicat, aquesta situació és força especial degut al canvi brusc de temperatura, factor sobre el que la nostra correcció de la desviació no actua.

En el cas de refredar el sensor (figura 3.13), els resultats semblen força ambigus, però en realitat no ho són. Si observem la forma de la ona, veiem com la correcció de les desviacions fa que sigui més constant, és a dir, que variï menys.

L'error que hi queda és degut als grans canvis de temperatura al que està exposat el sensor i a la impossibilitat d'aquest d'estabilitzar-se que, com ja hem dit, no és funció d'aquesta correcció arreglar, i no es donaran a la pràctica.

De fet, si observem el final de la gràfica, en les últimes 200 mostres, quan el sensor s'ha pogut estabilitzar, la senyal es troba pràcticament en zero.

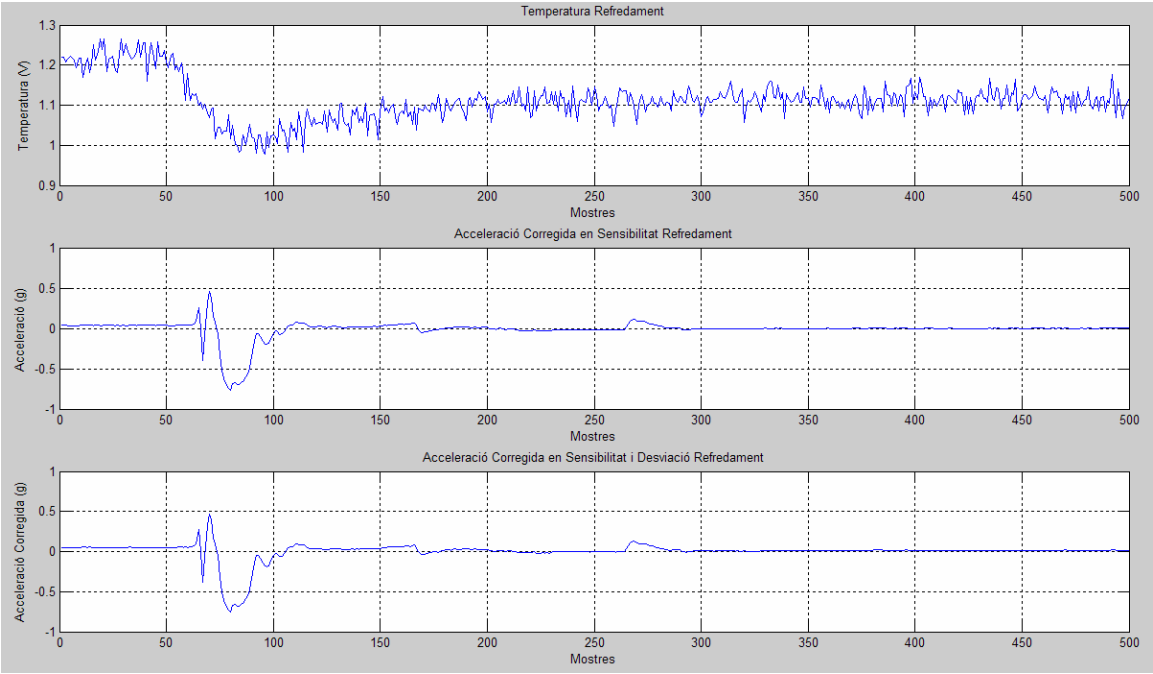

Figura 3.13: Desviació. Prova 2 Refredament

Finalment, i tot i no ser una mesura determinant per a detectar la qualitat d'una senyal, si fem la mitja d'ambdues senyals trobem que la que ha estat corregida en desviacions és més propera a zero.

Per evitar els errors comesos per els ràpids canvis de temperatura, i per adequar més la senyal d'acceleració a les acceleracions anteriors, hem d'aplicar un filtre a la senyal. L'objectiu del següent apartat és dissenyar aquest filtre.

3.3. Filtrat del Senyal

L'últim pas que ens queda per realitzar és el filtrat del senyal per a eliminar soroll de fons, irregularitats en les mostres i altres factors. Dissenyarem, doncs, un filtre adequat per al senyal que ens arribarà i l'aplicarem un cop haguem corregit la sensibilitat i l'offset.

Per les característiques que ha de tenir el filtre, està clar que haurà de ser un filtre pas baix, és a dir, un filtre que elimini aquella part del senyal que correspongui a altes freqüències. Això té avantatges i inconvenients:
Per una banda, la eliminació d'aquelles freqüències més altes ens suavitzarà el senyal, és a dir, tindrà canvis menys bruscos i pronunciats.
Per altra banda, farà que la resposta del sistema sigui més lenta, és a dir, tardarà més a reaccionar als canvis. A més a més, al passar per un filtre, la sortida serà inevitablement retardada.

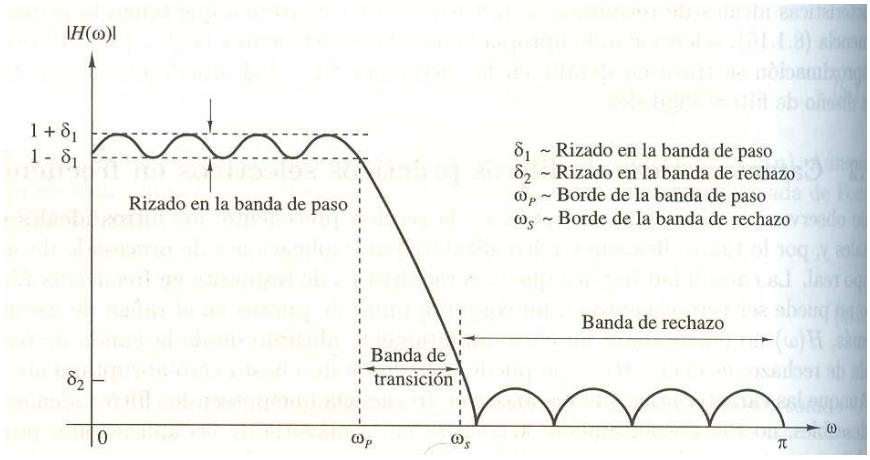

Figura 3.14: Característiques dels Filtres[1]

Si volem realitzar un filtre que millori el senyal però que no ens aporti un retard massa elevat ni ens elimini la capacitat de

[1] Imatge extreta de Tratamiento Digital de Señales, Proakis et al.

resposta, haurem d'establir un compromís entre la freqüència de pas i de mostreig, el grau del filtre i la qualitat del senyal.

3.3.1. Mètode de Disseny

Per al disseny del filtre treballarem amb l'eina MatLab, que disposa d'una aplicació per al disseny de filtres anomenada FDATool. Aquesta aplicació ens permetrà definir els paràmetres del nostre filtre, i obtenir directament els coeficients. Podríem consultar les coeficients en alguna bibliografia, però aquesta eina ens permetrà ajustar perfectament el filtre a les nostres necessitats.

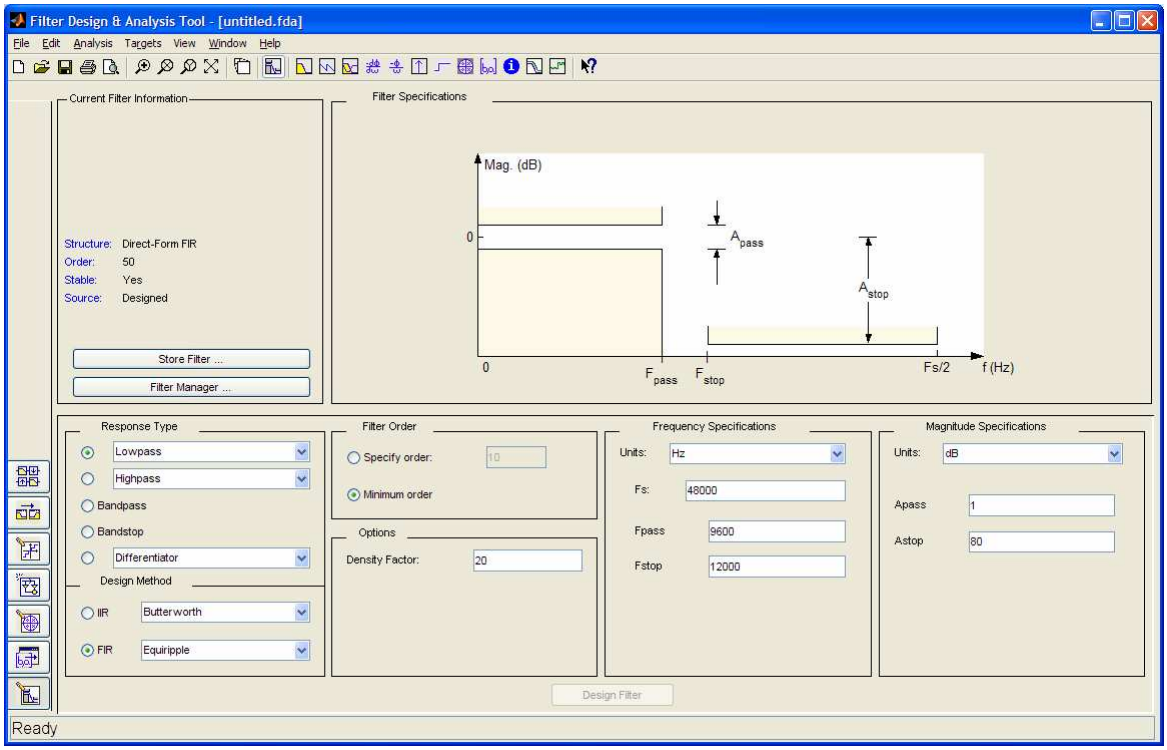

Figura 3.15: Eina FDATool de MatLab

Existeixen dos grans mètodes per al disseny de filtres, segons si el filtre és de resposta impulsional finita (Finitie Impulsion Response, FIR) o infinita (Infinite Impulsion Response, IIR). Això és

si la durada de la resposta a una entrada impulsional anirà disminuint fins a zero (FIR) o bé durarà infinitament (IIR).
Per motius d'implementació i facilitat de disseny, en quant a l'estabilitat del sistema, ens decantarem per un filtre FIR, és a dir, un filtre que al cap de suficient temps d'una entrada delta de Dirac, la resposta valdrà zero.

En general, un sistema (no només un filtre) FIR es defineix per la següent equació de diferencies:

$$y(n) = b_0 x(n) + b_1 x(n-1) + ... + b_{M-1} x(n-M+1) = \sum_{k=0}^{M-1} b_k \cdot x(n-k) \qquad (3.15)$$

On M-1 és el grau del sistema, és a dir, M serà el nombre de coeficients (o longitud) del sistema. I $\{b_k\}$ és el conjunt de coeficients del filtre. De la mateixa forma, podem expressar la sortida del sistema com la convolució de la resposta impulsional del sistema h(n) amb la senyal d'entrada,

$$y(n) = \sum_{k=0}^{M-1} h(k) \cdot x(n-k) \qquad (3.16)$$

De les dues equacions, idèntiques en la seva estructura, és fàcilment deduïble que,

$$h(k) = b_k, k = 0,1,...,M-1 \qquad (3.17)$$

També, podem expressar la funció de transferència del sistema en la transformada z:

$$H(z) = \sum_{k=0}^{M-1} b_k \cdot z^{-k} \qquad (3.18)$$

Com podem observar, el filtre FIR es tracta d'una suma de productes, on es treballa amb les M-1 últimes mostres per a treure la següent. Podem representar-lo amb el següent diagrama de blocs:

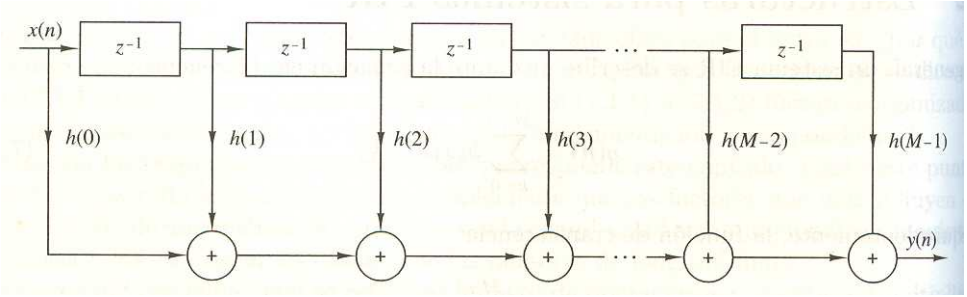

Figura 3.16: Diagrama Blocs Filtre FIR[1]

Una de les facilitats que ens aporten els filtres FIR és la possibilitat de tenir una fase lineal. La fase en un filtre està relacionada amb el retard que apareixerà en la senyal de sortida del filtre. Tenir una fase lineal significa que totes les freqüències seran desfasades de forma homogènia, així el retard que introdueix el filtre és constant per a totes les freqüències i de valor,

$$\frac{M-1}{2}\cdot T_0 \tag{3.19}$$

On T_0 és el període de mostreig, és a dir, l'invers de la freqüència de mostreig.

Per tant, per obtenir un retard equivalent a un nombre exacte de mostres, per a facilitar el futur tractament de les dades, haurem d'emprar una M imparell.

Si volem que el sistema FIR tingui fase lineal, haurem de fer que el sistema satisfaci les condicions de simetria o antisimetria:

$$h(n) = \pm h(M-1-n) \tag{3.20}$$

Per aquests sistemes la resposta impulsional està definida per la meitat del nombre de coeficients, M/2 per a M parell i (M-1)/2 per a M imparell.

En el cas d'un filtre pas baix, necessitem que es compleixi la condició de simetria amb M parell:

[1] Imatge extreta de Tratamiento Digital de Señales, Proakis et al.

$$h(n) = h(M-1-n) \tag{3.21}$$

Així, en el nostre cas, el filtre pas baix tindrà la resposta impulsional simètrica y amb un nombre de coeficients imparell; això últim ens garantirà que el retard introduït pel filtre sigui un nombre enter de mostres.

Finalment serà aquest algoritme el que haurem d'implementar en el nostre xip per tal de realitzar el filtrat del senyal. Tot i la simplicitat de les operacions individuals de cada fase, el conjunt de totes les operacions (segons el grau del filtre) pot arribar a consumir molta capacitat de còmput, per tant, haurem de controlar aquest fet.

Un cop vistes les característiques bàsiques del filtre FIR, haurem de definir aquells valors que ens ajudaran a fer que aquest sigui adequat per a la nostra aplicació. Es tracta de la freqüència de mostreig, de pas i el grau del filtre. Veurem aquestes definicions en els següents apartats.

3.3.2. Freqüències de Tall i de Mostreig

La tria de les freqüències de pas i de mostreig del filtre és força important en tant que condicionarà el rang de treball del filtre. Com veurem en el nostre cas, es tracta tant sols de posar sobre la taula les condicions de la nostra senyal.

El primer que farem és decidir la freqüència de mostreig. De fet, es tracta més d'una observació que d'una decisió, ara veurem perquè: com ja hem vist en la presentació del sensor, i en la introducció d'aquest mateix capítol, el sensor ens retorna el valor de

l'acceleració codificat en un tren de polsos, on la durada de la fase "alta" o de "on" indica l'acceleració.

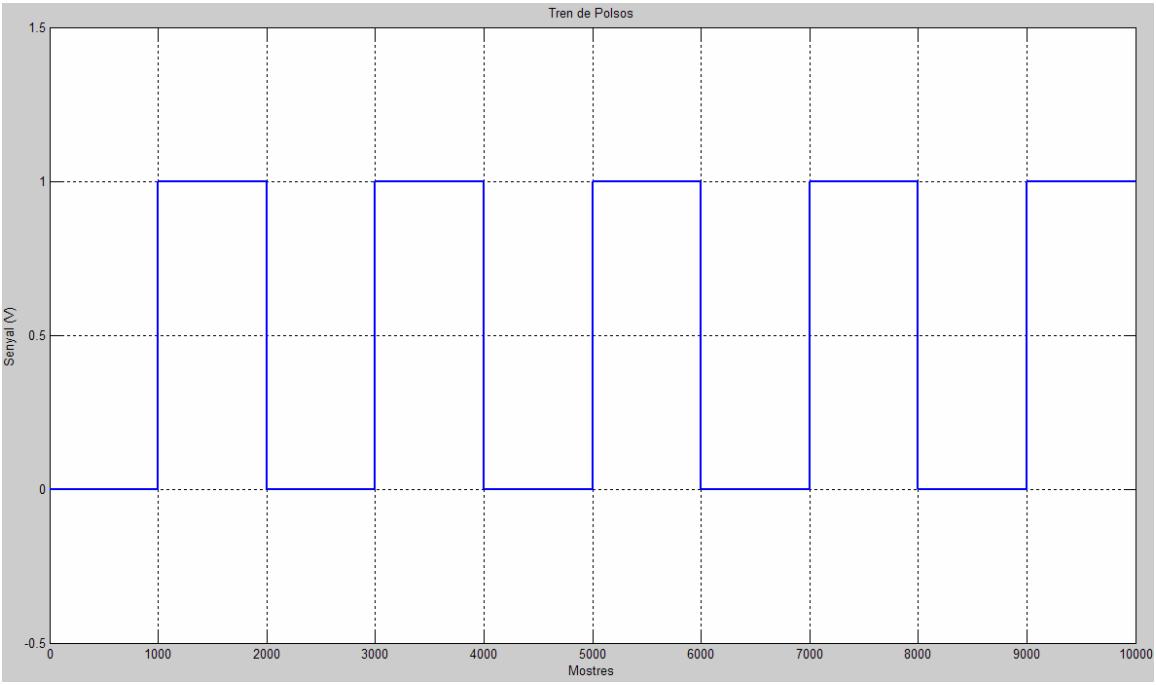

Figura 3.17: Tren de Polsos

El que no varia amb l'acceleració és el període d'aquest tren. Per conseqüència, la freqüència del tren tampoc varia, i aquesta és de 100Hz. Per tant, és el propi tren de polsos el que ens marca la freqüència de mostreig a 100Hz, o el que és el mateix, 100 mostres cada segon.

La freqüència de pas, en canvi, és un valor que no sembla tant evident, però si consultem les especificacions del sensor, també acabarem aclarint.

El fabricant ens indica que la freqüència màxima a la que treballa el sensor és 19Hz, amb un valor típic de 17 i un mínim de 15. Això significa que disposa d'un filtre pas baix intern que es troba centrat en 17Hz amb una banda de transició que va des de 15 a 19Hz. El que farem nosaltres és reduir lleugerament aquesta freqüència de tal forma, que els valors que obtinguem de freqüència no s'hagin

atenuat per la disminució progressiva de l'amplitud en la banda de transició.
Per aconseguir-ho fixarem la freqüència màxima de pas del nostre filtre a 15Hz. Com que no serà un filtre ideal, permetrà freqüències lleugerament més altes passar, però amb una amplitud inferior a 1, per tant amb poca repercussió.

En resum, dissenyarem el nostre filtre pas baix per a un treball amb mostres a 100Hz i una freqüència de tall de 15Hz. Ara, queda per definir el grau d'aquest filtre, que ens determinarà l'amplitud de la banda de transició.

3.3.3. Grau del Filtre FIR

La tria del grau del polinomi que representa el nostre filtre FIR és una decisió força important. D'aquesta tria dependrà directament que el nostre filtre sigui el més proper al filtre ideal possible.
És aquí on prenem un compromís entre la qualitat del filtre, el cost computacional del mateix i el retard acceptable pel sistema, en tant que un filtre de major qualitat representarà una major càrrega de còmput i també un major retard en obtenir la senyal filtrada.

La diferència entre, per exemple d'un filtre pas baix d'un grau 10 i un altre de grau 100 és molt gran. Podem observar aquesta diferència entre el filtre de la figura 3.18 i el de la figura 3.19, que són de grau 10 i 100, respectivament.
Mentre que en el primer veiem com la banda de transició és d'aproximadament 20Hz, en el segon és, amb prou feines de 4Hz. Evidentment, el cost d'implementar un i l'altre és també molt diferent.

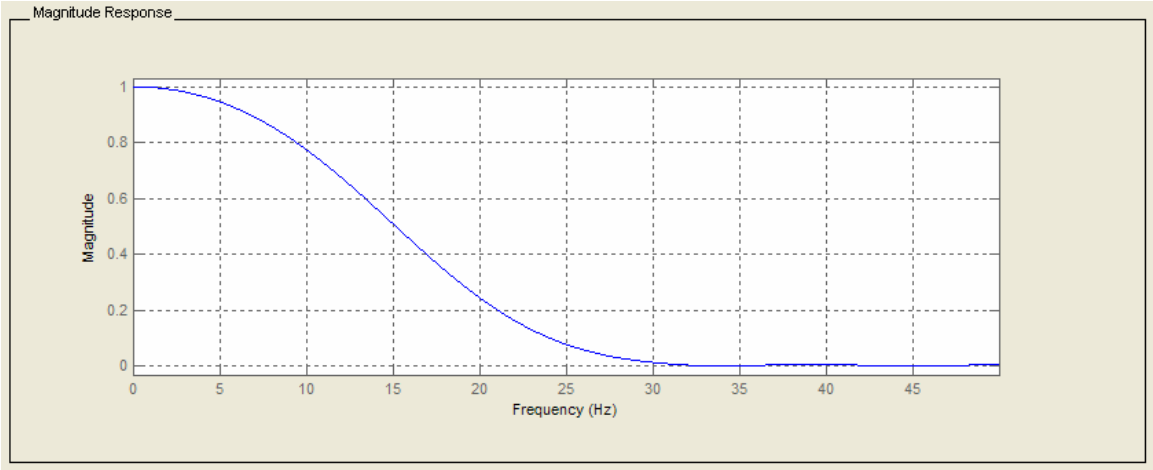

Figura 3.18: Filtre FIR de 10 coeficients

El primer filtre ens generarà un retard de 5 mostres i el segon de 50. Això, en temps, per exemple a 100Hz, representa un retard de 5ms i 50ms, respectivament.

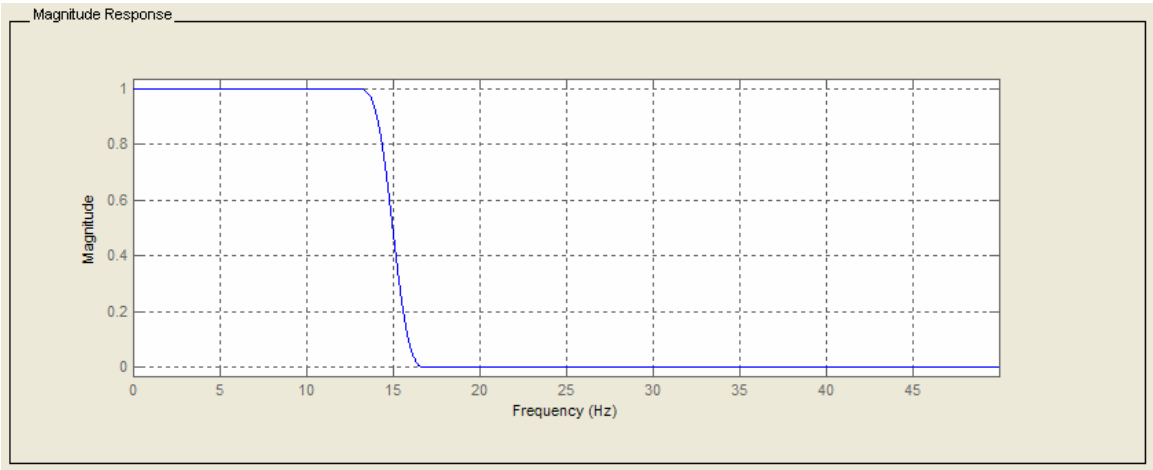

Figura 3.19: Filtre FIR de 100 coeficients

No és difícil establir que com més gran és el grau d'un filtre FIR, més s'apropa al filtre ideal. No obstant això, mai podrà arribar a ser un filtre ideal, ja que es tracta d'un truncament de la Sèrie de Fourier que aquest representa. Així doncs, tornem a la intenció d'establir un compromís entre precisió i còmput.

Ara és el moment d'estudiar les característiques del nostre processador i establir quin és el llindar màxim amb el que podem

treballar. Tenint 80Mhz de capacitat de còmput, amb instruccions de 32 bits, hem considerat que un grau més elevat de 80 podria causar certs problemes amb la interacció entre la captura de dades, de polsos i temperatura, i les operacions.

Més enllà d'aquest llindar, encara hem d'avaluar si un filtre tant acurat ens permetrà una gran millora, i per tant, val la pena. Si tenim en compte la qualitat del filtre intern del sensor, no és gaire pràctic carregar tant el processador si per una altra banda, hi apliquem un filtre dolent. Decidim, doncs, baixar a 50 el grau del filtre, de tal manera que la banda de transició acaba just al màxim de la freqüència màxima que ens podrà retornar el sensor.

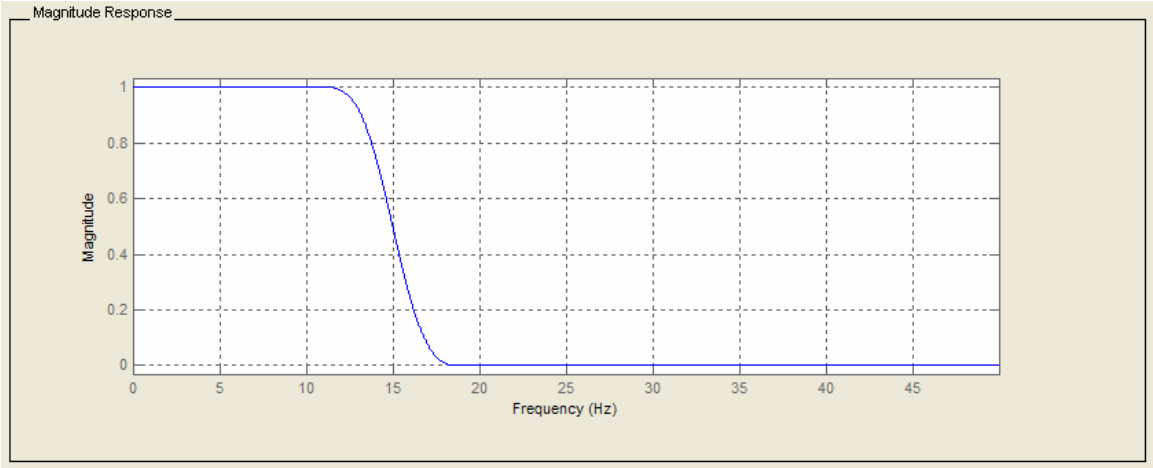

Figura 3.20: Resposta de Magnitud del Filtre

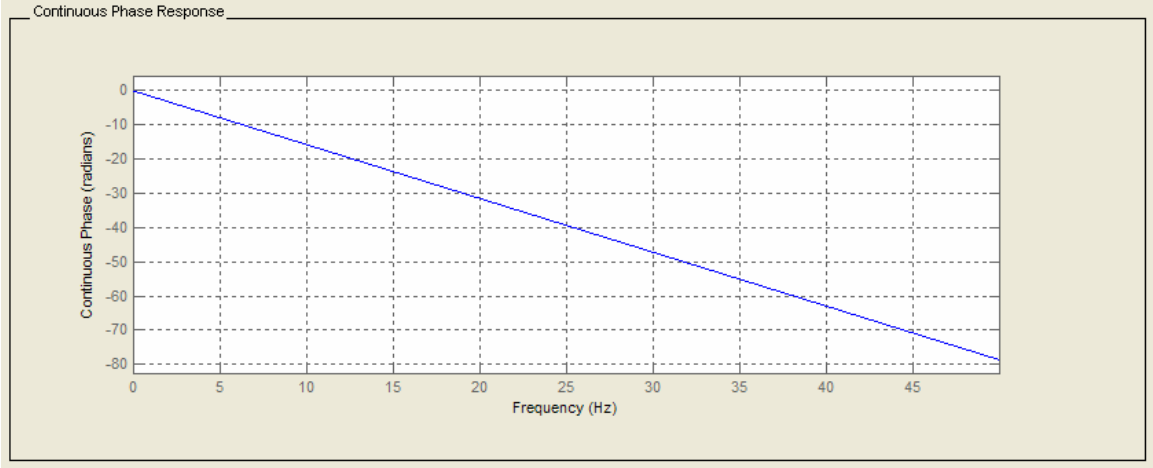

Figura 3.21: Fase del Filtre

Finalment ja disposem de totes les dades per a generar el filtre. Les introduïm a l'aplicació del MatLab i el dissenyem. A la figura 3.20 trobem el comportament en freqüència del filtre, mentre que a la figura 3.21 trobem la fase, que com podem comprovar és lineal.

3.3.4. Proves Realitzades

Un cop ja tenim generat el filtre, podem comprovar el seu funcionament directament amb MatLab i les dades que hem capturat del sensor.

La pròpia aplicació ens ofereix una eina per a filtrar dades amb els filtres que ens ha dissenyat. Tractarem amb dades que ja hem corregit en sensibilitat i a les que ja hem eliminat l'error per la desviació a 0g. Són les mateixes dades que hem vist en els altres procediments.

Els resultats en aquest cas per a la fase d'escalfament del sensor són les que podem veure en la figura 3.22, mentre que la figura 3.23 ens mostra el resultat de filtrar les dades de la fase de refredament.

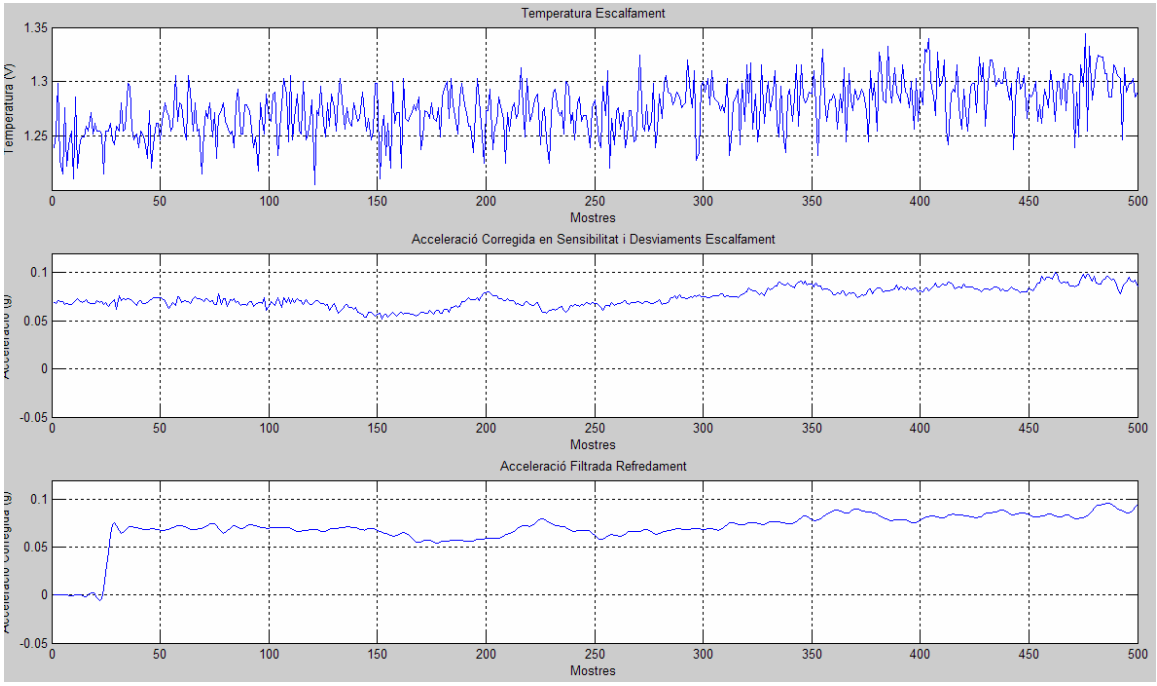

Figura 3.22: Filtre. Prova 1 Escalfament

Com podem veure, els resultats ens apareixen desfasats 25 mostres, és a dir, la meitat dels coeficients, o el que és el mateix, el grau del filtre.

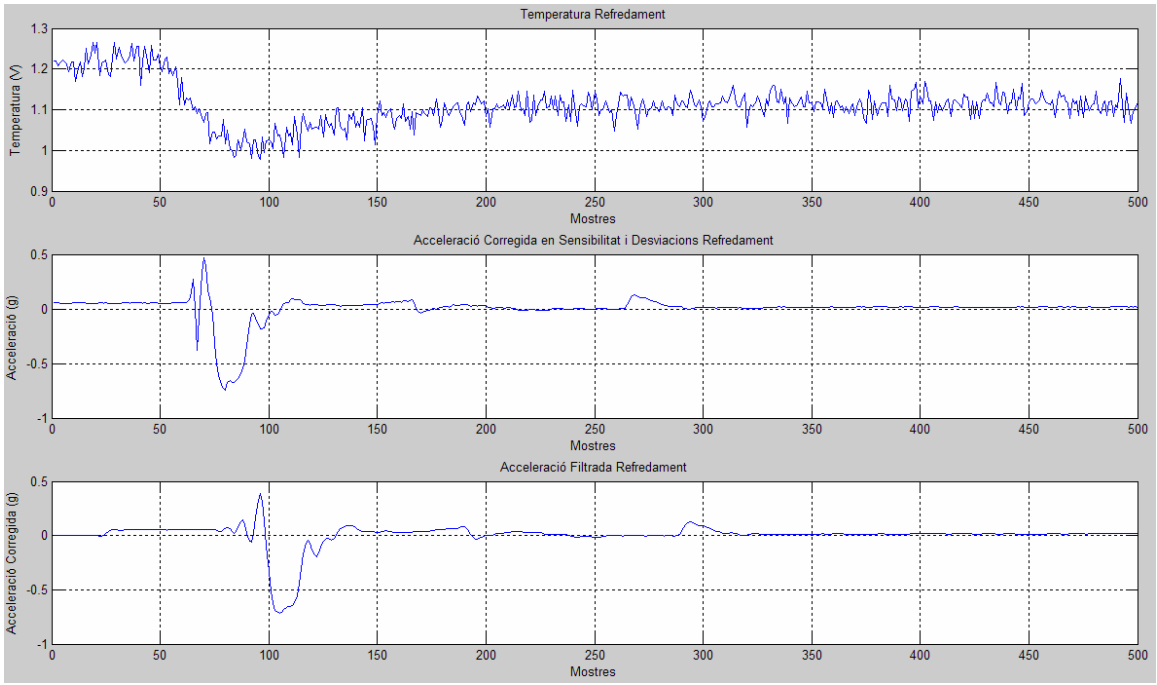

Figura 3.23: Filtre. Prova 2 Refredament

Altre cop hem realitzat la mitja dels valors obtinguts, que teòricament hauria de ser zero, i tot i que hem de recordar que no és una mesura estadística vàlida, ja que els valors positius i els negatius es poden arribar a compensar, tenim en ambdós casos mitjanes més properes a zero.

Finalment, destacar que en ambdós caos la senyal apareix amb molt menys soroll, sense tanta rugositat i oscil·lació de valors ocasionats per el soroll de fons, sinó que és més suau, més contínua, que era l'objectiu d'aquest filtre.

Aquest és ja l'últim pas del procés de tractament del senyal. Aquestes dades ja són considerades com a definitives, i per tant són les que s'enviaran a l'encarregat de gestionar-les. En els següent apartat veurem com implementar aquests procediments en el nostre processador i enviar les dades resultants.

4. ColdFire MCF5213

L'objectiu d'incorporar un apartat per a descriure el microprocessador i aquells mòduls dels que farem ús és crear un coneixement comú sobre el microcontrolador per tal que el lector d'aquest document tingui clars els conceptes relacionats amb el xip i pugui seguir les explicacions futures.

Una gran part del temps invertit en aquest projecte ha estat encarat a la comprensió del funcionament intern del MCF5213 i els seus diversos mòduls.
És essencial entendre de forma global com tractar un dispositiu d'aquestes característiques abans de posar-s'hi a treballar.

Els programadors estan acostumats a treballar sobre un llenguatge, sense haver de pensar en un hardware que hi ha radere. Al desenvolupar aplicacions embedides en xips, s'ha d'aprendre a pensar en els mòduls que tenim dins del nostre microprocessador, en els registres, en les rutines, en les interrupcions, en senyals, en temporitzadors i un llarg etcètera.

Esperem que aquesta explicació resulti aclaridora per al lector, que sempre cal que recordi que, en cas de necessitar un coneixement més explícit o detallat, pot consultar el full de dades del fabricant.

Cal tenir en compte que aquest microcontrolador es troba dins la família MCF521X, que implementa V2 de la arquitectura ColdFire. Això fa que implementi moltes funcions que nosaltres potser no utilitzarem, però que són pròpies d'aquesta arquitectura.

El primer que farem és enumerar les característiques principals del xip. No estem parlant ja de la família MCF521X, sinó del xip MCF5213 en concret, que és el que hem escollit:

- Freqüència de rellotge: 80MHz
- Rendiment: fins a 76 MIPS
- 32 KBytes de RAM
- 256 KBytes de memòria flash
- 3 UARTs
- Controlador FlexCAN 2.0B
- Controlador I^2C
- Convertidor Analògic-Digital de 12 bits
- Queued Serial Peripherial Interface (QSPI)
- 4 DMA de 32 bits
- 2 Periodic Interrupt Timers (PITs) de 16 bits
- Gestor d'Interrupcions amb 63 entrades

També cal exposar els principals mòduls que han estat inclosos en aquest xip. Podem trobar alguns mòduls de control que no ofereixen cap recurs a l'usuari que han estat ignorats:

- System Control Module
- General Purpose I/O Module
- Interrupt Controller Module
- Edge Port Module
- DMA Module
- Programmable Interrupt Timer Module
- General Purpose Timer Module
- QSPI Module
- UART Module
- I^2C Module
- Analog-to-Digital Converter Module
- Pulse Width Modulation Module
- FlexCAN Module

- Debug Module

A banda de definir els mòduls, el fabricant també ens indica com és relacionen entre ells. Concretament ens ofereix l'esquema de la figura 4.1 que mirat detalladament resulta força aclaridor.
De fet, el propi entorn de programació CodeWarrior ens ofereix una guia força bona de l'estructura interna del MCF5213 a través d'una representació física del xip amb els mòduls dels que disposem reflectits. Hem capturat aquesta representació en la figura 4.2.

Evidentment, no utilitzarem tots els mòduls que ens ofereix aquest xip, sinó que escollirem aquells més adequats per a la nostra aplicació. Després de realitzar aquesta tria, estudiarem els escollits amb més atenció.

El primer mòdul que hem de pensar és aquell que ens permetrà obtenir la senyal d'acceleració del sensor. Com ja hem vist, el sensor ens retorna la senyal d'acceleració codificada en l'ample de pols d'un tren de polsos. Així, necessitem un mòdul que ens permeti comptar el temps que transcorre entre el canvi d'estat de baix a alt, i el canvi d'estat d'alt a baix.

Aquest mòdul haurà de ser capaç de detectar canvis d'estat en un senyal i portar el recompte del temps transcorregut. El mòdul ideal per a fer aquesta tasca és el General Purpose Timer (GPT) Module.

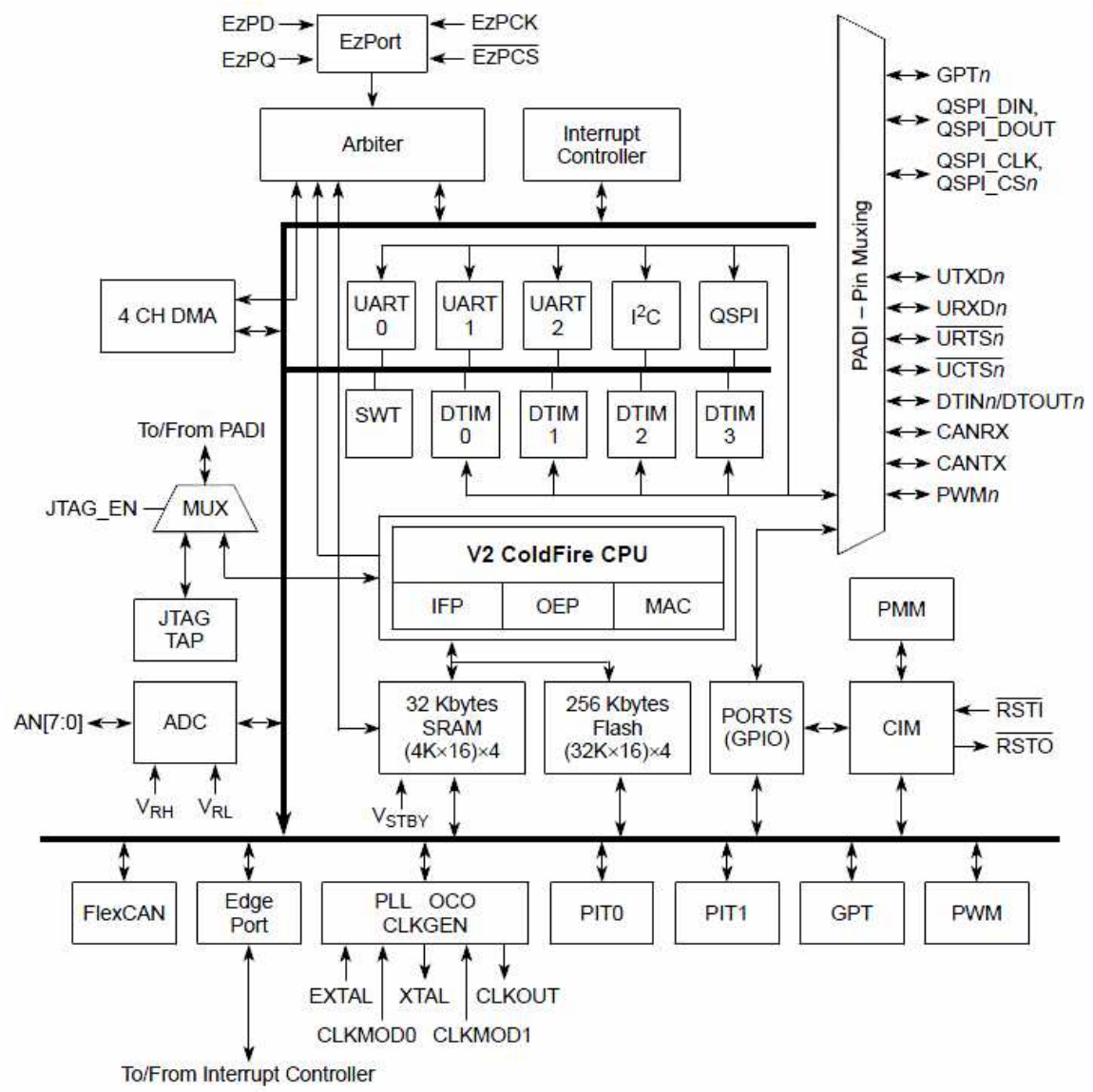

Figura 4.1: Esquema del MCF5213

Disposem alhora d'un altre mòdul que ens permet portar el compte del temps, el DMA Module, que disposa d'una major resolució per hardware, i un temporitzador configurable independent, però no ofereix la possibilitat de detectar canvis en el senyal de forma directa. Havíem pensat utilitzar el GPT per a capturar els canvis en el senyal, i el DMA per a portar el compte del temps transcorregut, però amb la possibilitat de tractar el *overflow* del GPT a través d'interrupcions software, el mòdul DMA no ens aporta cap benefici.

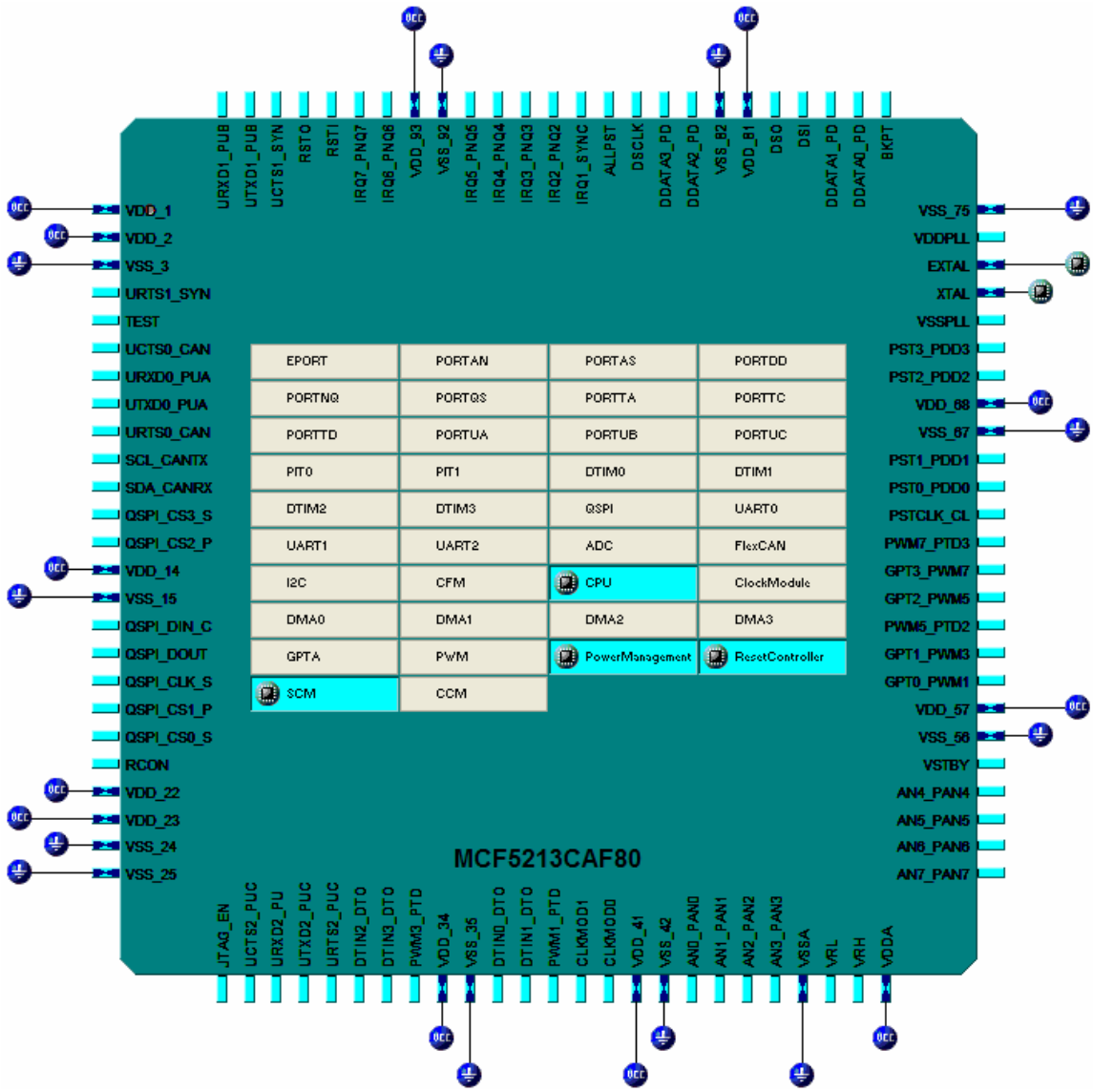

Figura 4.2: Mòduls del MCF5213

Sembla força evident que per a la captura de la senyal de temperatura hem d'escollir el mòdul Analog-to-Digital Converter (ADC). Aquest mòduls ens permetrà obtenir el valor del voltatge de la senyal en un instant determinat. Donat que disposem de 8 canals, podríem mostrejar fins a 8 senyals diferents. En el nostre cas només seran dos.

L'últim mòdul que utilitzarem serà el mòdul de comunicació I^2C. Com hem vist en la proposta de solució del problema, el bus general de dades serà un bus I^2C, per tant, es fa indispensable

utilitzar aquest mòdul per a enviar les dades resultants al gestor del sistema.

En els següents apartats veurem una explicació més detallada d'aquests tres mòduls:

- Analog-to-Digital Converter (ADC)
- General Purpose Timer (GPT)
- I^2C

Hem decidit classificar l'explicació en 4 apartats. En el primer explicarem les característiques bàsiques del mòdul. En el segon tractarem el funcionament general del dispositiu, mentre que en el tercer ens centrarem en la configuració aplicada al nostre cas en concret. Finalment explicarem com accedir a les dades o funcionalitats de cada mòdul.

El lector podrà veure com varia força la forma d'actuar segons si ho realitzem sense ajuda de l'entorn de programació CodeWarrior (CW), si ho realitzem amb l'ajuda de configuració d'aquest, o bé utilitzem l'aplicació Processor Expert dins del propi CW per tal de configurar i accedir als serveis de cada mòdul.

4.1. Analog-To-Digital Converter (ADC)

El mòdul ADC captura el voltatge de la senyal que arriba al microcontrolador per una pota concreta i l'avalua. Direm que l'avalua per que no ens retorna un valor exacte de voltatge, sinó un valor relatiu als voltatges de referència dels que disposa el microprocessador o bé li proporciona l'usuari.

4.1.1. Característiques ADC

Aquest mòdul te unes característiques força avançades per tractar-se d'una part d'un microcontrolador i no d'un xip exclusivament dedicat a la conversió de senyals analògiques a digitals. Entre moltes altres, volem destacar les següents:

- Resolució de 12 bits
- 8 canals de mesura
- Captura simultània de 2 valors
- Mètodes de captura configurables
- Fins a 1'66 milions de mostres per segon
- Voltatges de referència interns o definits per l'usuari
- Resultat amb o sense signe

A part d'aquestes característiques, una que resulta de vital importància i que per això tractarem més a fons en el següent apartat és la possibilitat de tenir diversos modes d'operació, segons tres paràmetres.

4.1.2. Funcionament ADC

El procés de conversió d'analògic a digital és força senzill. En realitat es tracta d'una comparació i no d'una conversió. El que fa el microcontrolador és comparar la senyal d'entrada amb un voltatge de referència i ens diu si és igual al voltatge superior (4095), a l'inferior (0) o bé un valor entre aquests dos. Si el que volem és conèixer el voltatge real, hem de conèixer els valors dels voltatges de referència i obtenir la magnitud d'una unitat en aquesta escala:

$$\frac{V_{REFH} - V_{REFL}}{4095} \tag{4.1}$$

On V_{REFH} és el voltatge de referència superior i V_{REFL} és el voltatge de referència inferior.

Tot i que els valors de referència seran els mateixos per als dos, aquest mòdul disposa de dos unitats de conversió, és a dir, podem convertir dues senyals alhora, obtenint resultats simultanis en el temps. El diagrama de blocs de la figura 4.3 resulta molt aclaridor.

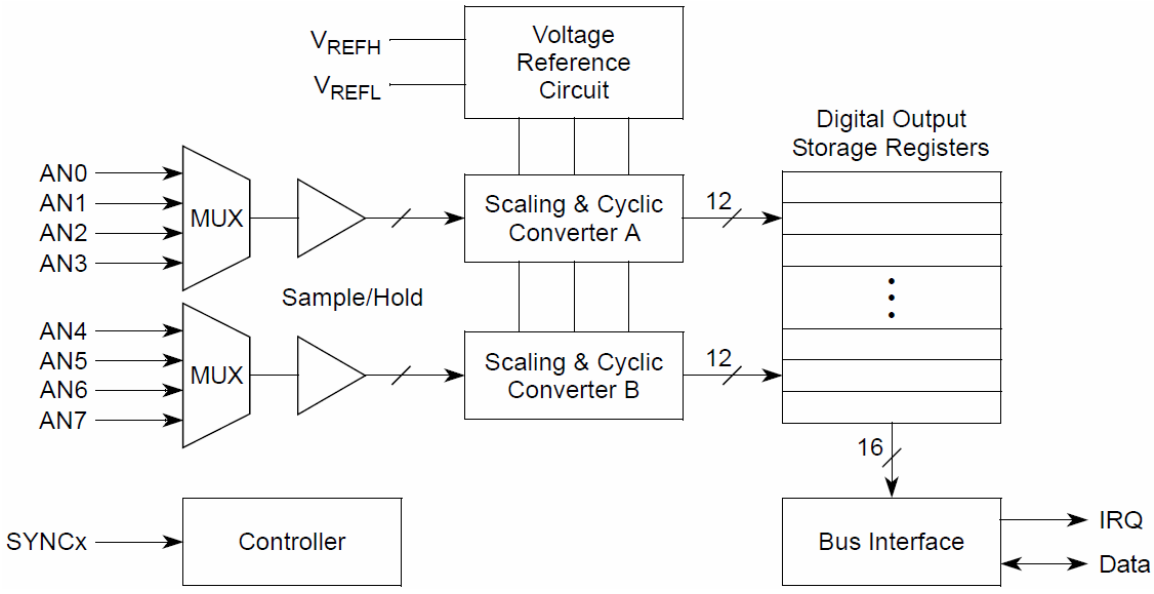

Figura 4.3: Diagrama de Blocs ADC

Com podem apreciar en el diagrama de blocs, disposem d'un multiplexor que ens permet realitzar la selecció del canal que volem analitzar, un amplificador que augmenta la senyal i finalment un convertidor cíclic per a obtenir el valor comparant-lo amb els voltatges de referència. Finalment, guardarem el resultat al registre corresponent.

A part del funcionament general del procés, el mòdul ens permet configurar la forma en que es prenen els valors. El fabricant anomena modes d'operació les possibles combinacions dels tres valors de configuració:

- **Mètode d'entrada**: es tracta del mode en que es tractaran els canals. Tenim dues opcions:
    - *Single-Ended*: es tracta de capturar únicament el valor del voltatge del canal.
    - *Diferential*: realitza una resta entre el valor del voltatge d'un canal amb un altre, tenint per als 8 canals, 4 resultats.
- **Mètode de captura**: consisteix en definir com s'analitzaran tots els canals, quin procediment seguirem per a retornar la conversió de totes les entrades:
    - *Sequential*: recorrerem tots els canals de forma seqüencial, començant per el canal 0 i acabant per el canal 7.
    - *Parallel*: avaluarem els canals 2 a dos, aprofitant que tenim 2 unitats de conversió analògico-digital. Tardarem, doncs, la meitat del temps que amb el mode seqüencial.
- **Iniciació de la captura**: és el mecanisme que iniciarà el procés de captura o bé el procés d'una captura, segons veurem:

- *Once*: es tracta de només realitzar una captura. Es farà la conversió de tots els canals amb el mètode de captura que correspongui i s'aturarà el mòdul.
- *Loop*: el ADC estarà fent conversions sense aturar-se, ja sigui de forma paral·lela com seqüencial.
- *Triggered*: una senyal o succés extern, capturat d'alguna forma per el microcontrolador, serà el que determinarà el moment de realitzar una conversió. Aquesta opció serveix per a sincronitzar les captures amb una senyal externa.

Les possibles combinacions dels diversos valors que pot prendre cada variable donen lloc a configuracions del mode d'operació, per exemple, una configuració "*single ended* – *seqüencial* – *once*" prendria una mostra de cada un dels canals de forma seqüencial només una vegada, mentre que una configuració "*diferencial* – *parallel* – *loop*" prendria contínuament mostres i les restaria, una de cada unitat de conversió.

4.1.3. Configuració ADC

En aquest apartat descriurem per una banda la configuració que nosaltres utilitzarem i per altra, el mecanisme que tenim per a aplicar-la.

En el nostre cas utilitzarem una captura *Single-Ended* de forma paral·lela i en mode *loop*, d'aquesta forma sempre tindrem el resultat de la última temperatura disponible, però sense consumir temps de còmput del processador. Podem veure un esquema de funcionament d'aquest mètode a la figura 4.4.

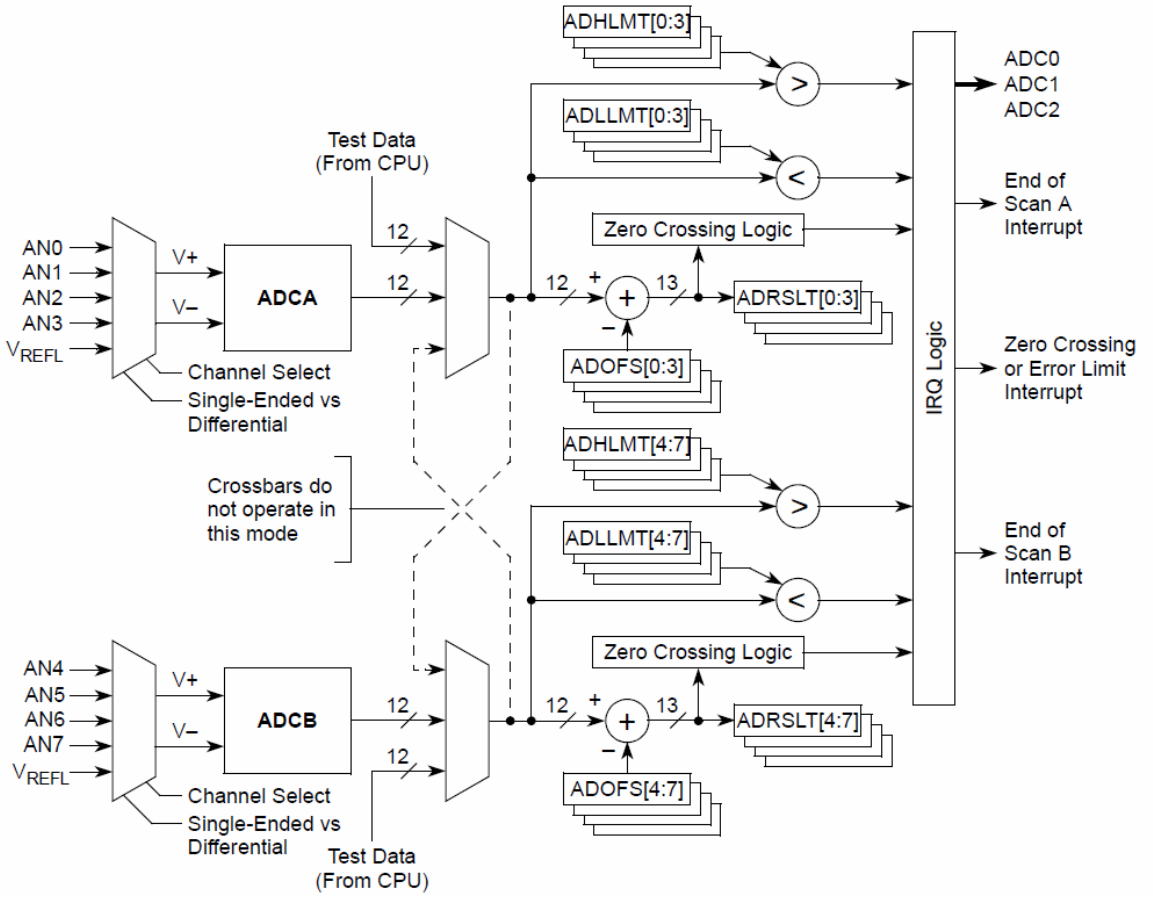

Figura 4.4: Esquema de funcionament ADC

Utilitzarem el mode simultani per que utilitzarem el mateix processador per a llegir la temperatura de dos sensors d'acceleració que, entre els dos, ens descriuran l'acceleració en els 3 eixos. Per això utilitzarem el canal 0 (del convertidor A) i el canal 4 (del convertidor B), ja que són els primers de cada convertidor, per tant el temps encara serà menor al no haver de passar per altres canals.

Per a aplicar la configuració desitjada al xip tenim diverses opcions. La primera i més rudimentària és fer-ho a través de la manipulació directa dels registres de configuració.

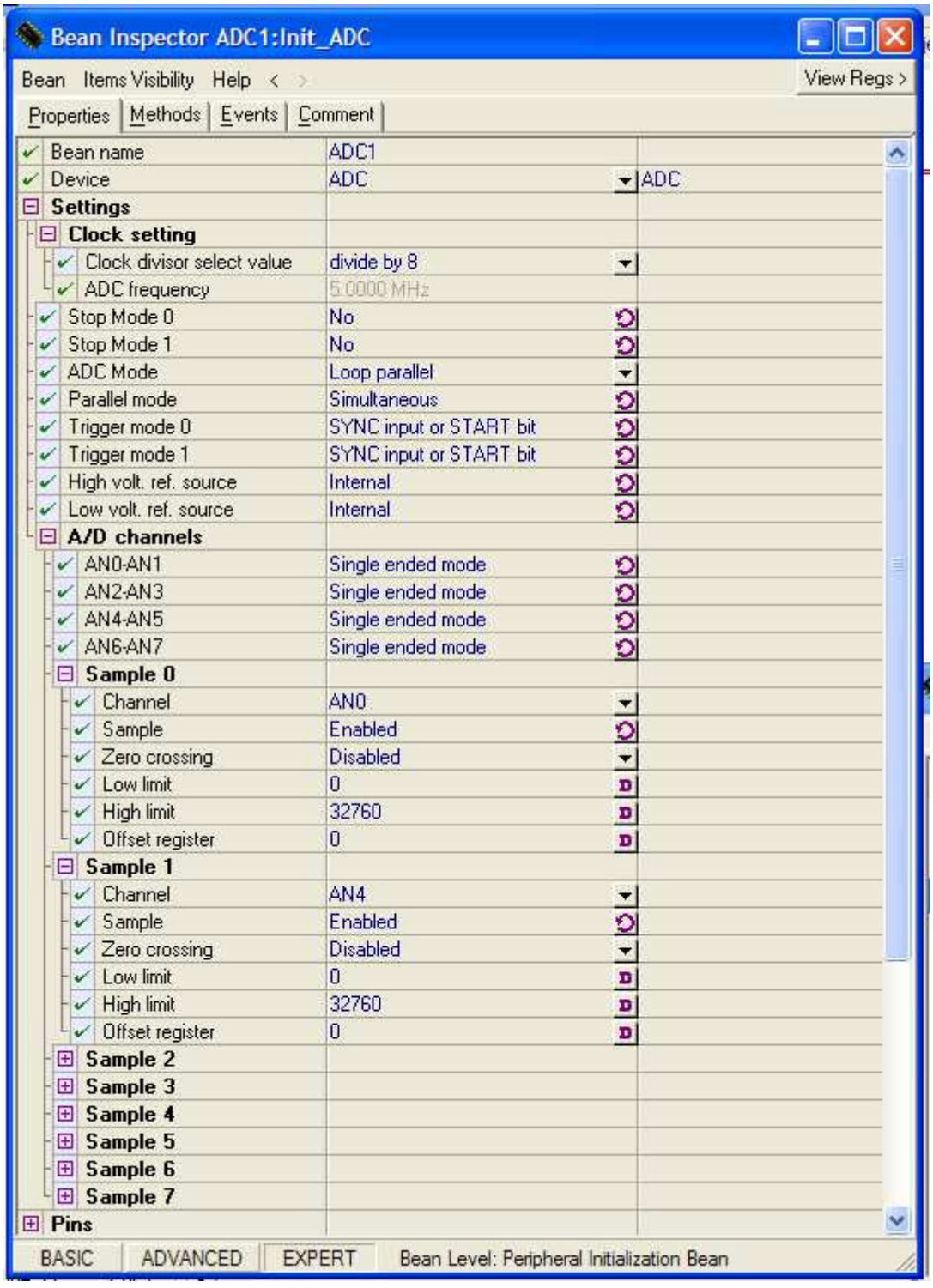

Figura 4.5: Configuració ADC I

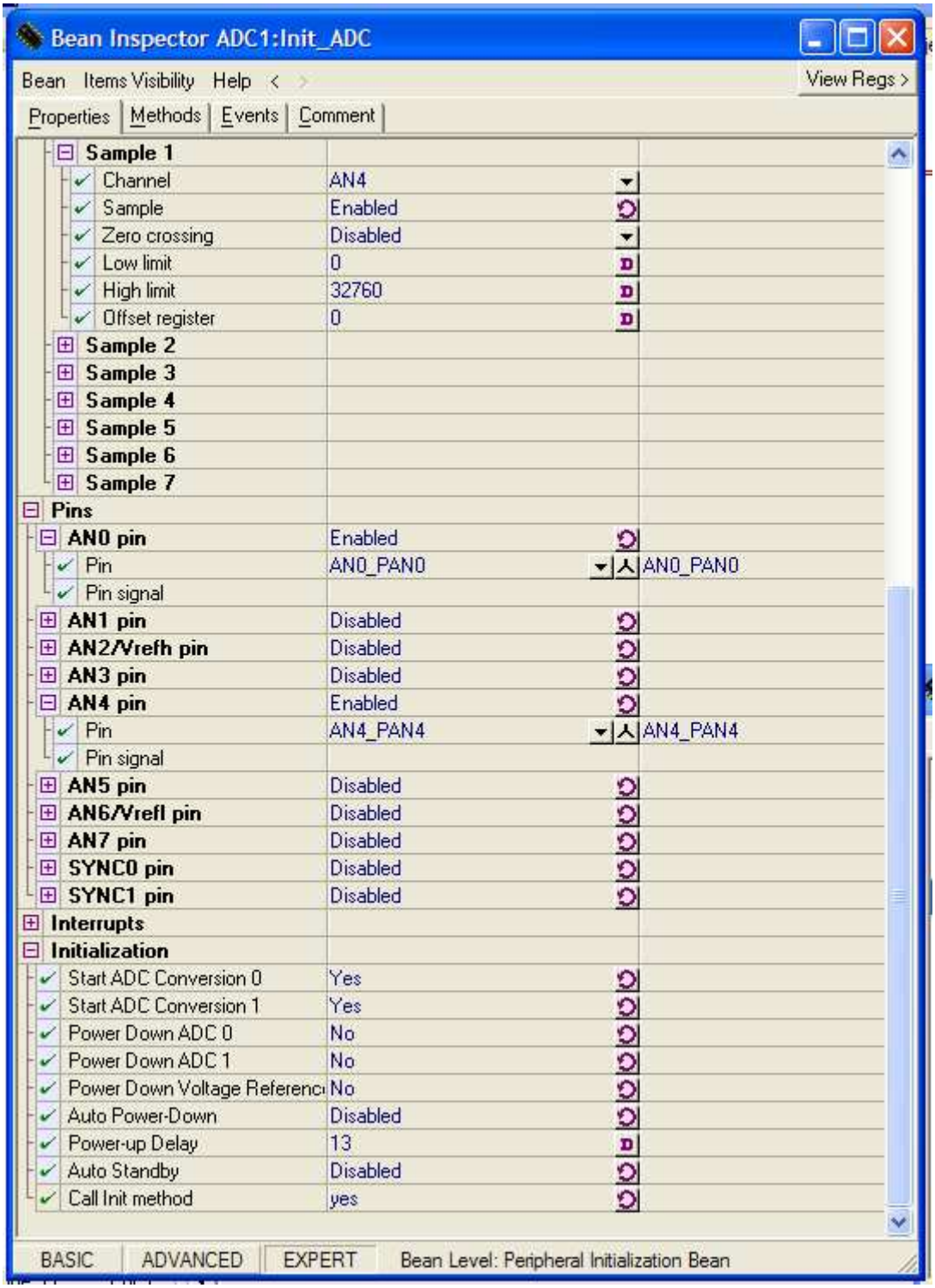

Figura 4.6: Configuració ADC II

Una altra opció és utilitzar una aplicació, dins del propi CodeWarrior, que ens generi el codi adequat per a configurar el processador segons els paràmetres que nosaltres li hem passat.

La última és utilitzar el Processor Expert, que és un programa que forma part de l'entorn CodeWarrior, que no només ens permet realitzar la configuració del processador sinó que també ens ofereix unes llibreries per a tenir accés a les funcions del mòdul a alt nivell.

Tot i que en altres ocasions ens decantarem per l'ús del Processor Expert i les seves funcions, en aquest cas només emprarem la petita aplicació dins del propi CodeWarrior que ens genera el codi de configuració del xip (figures 4.5 i 4.6).

4.1.4. Utilització ADC

Finalment, només ens queda detallar com recollirem la informació resultant del mòdul ADC. Donat que hem decidit no treballar amb les llibreries que ens ofereix el Processor Expert, si més no, no per al mòdul ADC, haurem d'accedir directament als registres de resultats.

Existeix un registre de resultat per a cada canal del ADC, per tant, tenim 8 registres de resultats, del 0 al 7. Com que hem decidit treballar amb els canals 0 i 4, haurem de consultar els registres ADRSLT0 i ADRSLT4.

IPSBAR 0x19_0012 (ADRSLT0) Access: read/write
Offsets: 0x19_0014 (ADRSLT1)
0x19_0016 (ADRSLT2)
0x19_0018 (ADRSLT3)
0x19_001A (ADRSLT4)
0x19_001C (ADRSLT5)
0x19_001E (ADRSLT6)
0x19_0020 (ADRSLT7)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SEXT				RSLT									0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 4.7: Registre ADRSLTn

La figura 4.7 ens mostra l'estructura d'un registre de resultats i les adreces de tots els registres. A la pràctica, només hem de conèixer el nom del registre i realitzar la lectura. Com veiem el resultat s'haurà de desplaçar a la dreta ja que en el registre es troba desplaçat a l'esquerra.

4.2. General Purpose Timer (GPT)

Podem parlar del General Purpose Timer (GPT) com d'un temporitzador que ens marca el temps que ha passat des de la seva posada en marxa. D'aquesta manera, si guardem el valor del comptador en dos instants de temps separats, podrem saber el temps que ha transcorregut si restem aquests dos valors.

4.2.1. Característiques GPT

Essencialment podríem dir que el GPT és un comptador de cicles de rellotge de la CPU amb registres de 16 bits. Bàsicament, el General Purpose Timer ens ofereix les següents possibilitats, si més no, a nivell general:

- Comptador de 16 bits
- 4 canals
- Freqüència programable
- Possibilitat de tenir rellotge extern
- Control del *overflow*
- Comptador d'events externs

Aquesta última opció representa un gran avantatge a l'hora de treballar amb la captura de senyals, ja que podem configurar-lo per a que ens notifiqui el canvi d'estat d'un senyal. Veurem aquesta funcionalitat amb més detall al llarg d'aquest document.

4.2.2. Funcionament GPT

El GPT consisteix en un comptador de 16 bits amb 4 canals. Utilitzant el rellotge intern del sistema i escalat segons un factor

escollit (en tenim 7 possibles), el que fa el GPT és comptar els períodes de rellotge que passen i guardar aquest recompte en un dels 4 registres possibles (un per canal). Podem veure un esquema de funcionament en la figura 4.8).

En casos en els que el comptador tingui un escalar molt baix, i per tant la seva freqüència sigui molt alta, haurem de controlar que no haguem superat la capacitat del registre, o bé, al superar-la, tenir-ho en compte i corregir-ho en els càlculs. Per a facilitar aquesta feina el microcontrolador llançarà una interrupció cada vegada que superem la capacitat del registre.

Com ja hem dit en l'apartat 4.2.1, una de les característiques clau és el fet que ens permeti actuar com a comptador de successos externs. Això vol dir que pot reaccionar als canvis en un senyal que arriba al microprocessador. Tot i el genèric de l'expressió, aquesta funcionalitat permet moltes aplicacions, sobretot per descodificar senyals.

Per a controlar els canvis en el senyal, el processador ens ofereix tres possibilitats per a detectar-los. A més a més, podem canviar la nostra tria al llarg de l'execució de l'aplicació:

- La senyal passa d'estat baix a estat alt (flanc de pujada)
- La senyal passa d'estat alt a baix (flanc de baixada)
- La senyal canvia d'estat (ambdós flancs)

En els tres casos, i això és el més important, ens llançarà una interrupció indicant que hi ha hagut un canvi d'estat. Serà en la rutina de tractament d'aquesta interrupció on posarem el nostre codi.

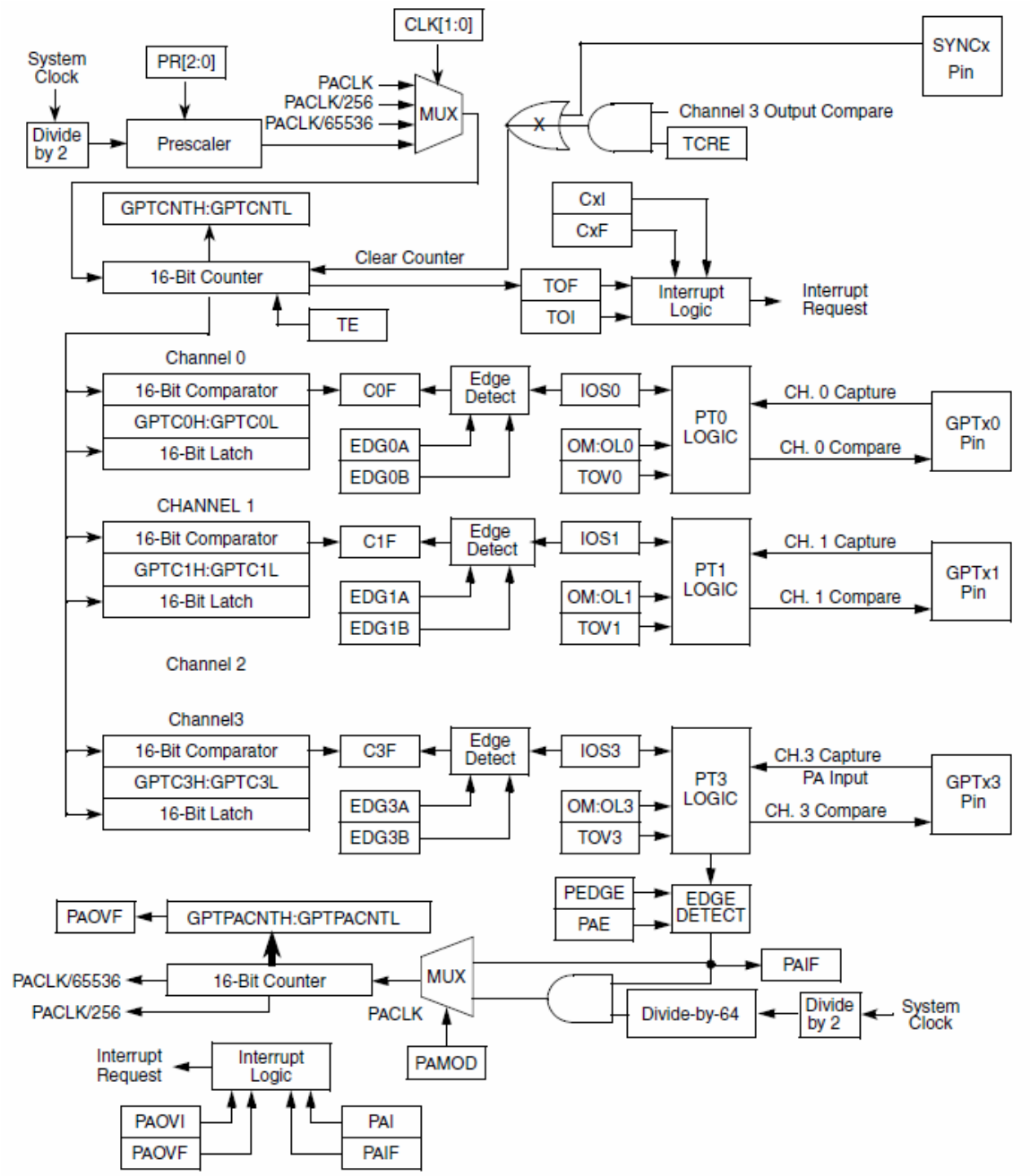

Figura 4.8: Diagrama de Blocs GPT

Fixem-nos ara en una possible aplicació en la que vulguem capturar l'ample de pols d'un tren de polsos. En aquest cas el primer que faríem és activar la interrupció per flanc de pujada. En quan saltés la interrupció sabríem que ens trobem en la part "alta" o de "on" del tren, guardaríem el valor del GPT en una variable i canviaríem la situació que activa la interrupció al flanc de baixada. La següent vegada que saltés la interrupció ens trobaríem en el

flanc de baixada, per tant podríem obtenir el temps de "on" senzillament restant d'aquest segon instant el valor del primer. Caldria també tenir en compte els possibles *overflows*, però resultaria tant senzill com sumar al resultat el nombre d'*overflows* produït multiplicat per el valor del *overflow*.

També hem de parlar de la freqüència del GPT. Com sabem, el mòdul ens ofereix diversos escalars que ens permeten reduir la freqüència d'execució en relació amb la freqüència general del dispositiu. La freqüència del GPT serà el resultat d'aquesta senzilla operació:

$$f_{GPT} = \frac{f_{processador}}{escalar} \tag{4.2}$$

L'elecció de la freqüència adequada, doncs, significa escollir l'escalar adequat. Però com ho fem això? Bé, la resposta és senzilla si ens fixem en els diferents efectes que pot produir l'elecció de l'escalar.

Si triem un escalar gran, la freqüència del GTP serà menor, per tant això ens permetrà contar events que succeeixin distants en el temps, però amb menor precisió, ja que una unitat del GPT equivaldrà a moltes unitats de temps.

Per contra, si triem un escalar petit, la freqüència del GPT serà més gran, i per tant això ens permetrà comptar successos més pròxims en el temps però amb major precisió. Ara bé, si tenim en compte que podem treballar amb el concepte d'*overflow*, i tenir-lo en compte, una altra freqüència del GPT no significa haver de tenir els successos pròxims, sinó que es pot fer ús de l'*overflow* i tenir-los separats en el temps.

Havent vist ja el principi de funcionament del GTP i les funcions de control de l'*overflow* així com el canvi del flanc de

detecció, ens queda detallar quina serà la configuració que utilitzarem nosaltres.

4.2.3. Configuració GPT

En el nostre cas haurem de llegir de 2 sensors d'acceleració, per tant estem parlant de llegir fins a 4 trens de polsos. Evidentment dos d'aquests trens ens donaran dades de la mateixa dimensió, però les capturarem i després ja decidirem què fem amb aquestes dades.

La tècnica de capturar dades és la que hem vist en l'apartat anterior com a exemple d'aplicació d'aquest mòdul GPT. Consistirà en primer activa la interrupció de canvi d'estat per flanc de pujada, i en la gestió d'aquesta interrupció canviar el flanc al de baixada. En la gestió del flanc de baixada el tornarem a canviar a pujada, i així successivament.
També haurem de tenir en compte l'*overflow*. En l'apartat del disseny de l'aplicació ens centrarem en els problemes que puguin aparèixer a l'hora de produir el nostre programa.

Per a realitzar la configuració del mòdul de GPT utilitzarem l'eina Processor Expert que ens ofereix el CodeWarrior (figura 4.9). Activarem un mòdul per cada canal que volem llegir, per tant, haurem de realitzar 4 configuracions. Tot i que les configuracions genèriques del comptador (freqüència, etc.) són comuns per als 4, haurem de treballar amb els 4 mòduls per a tenir-los actius.

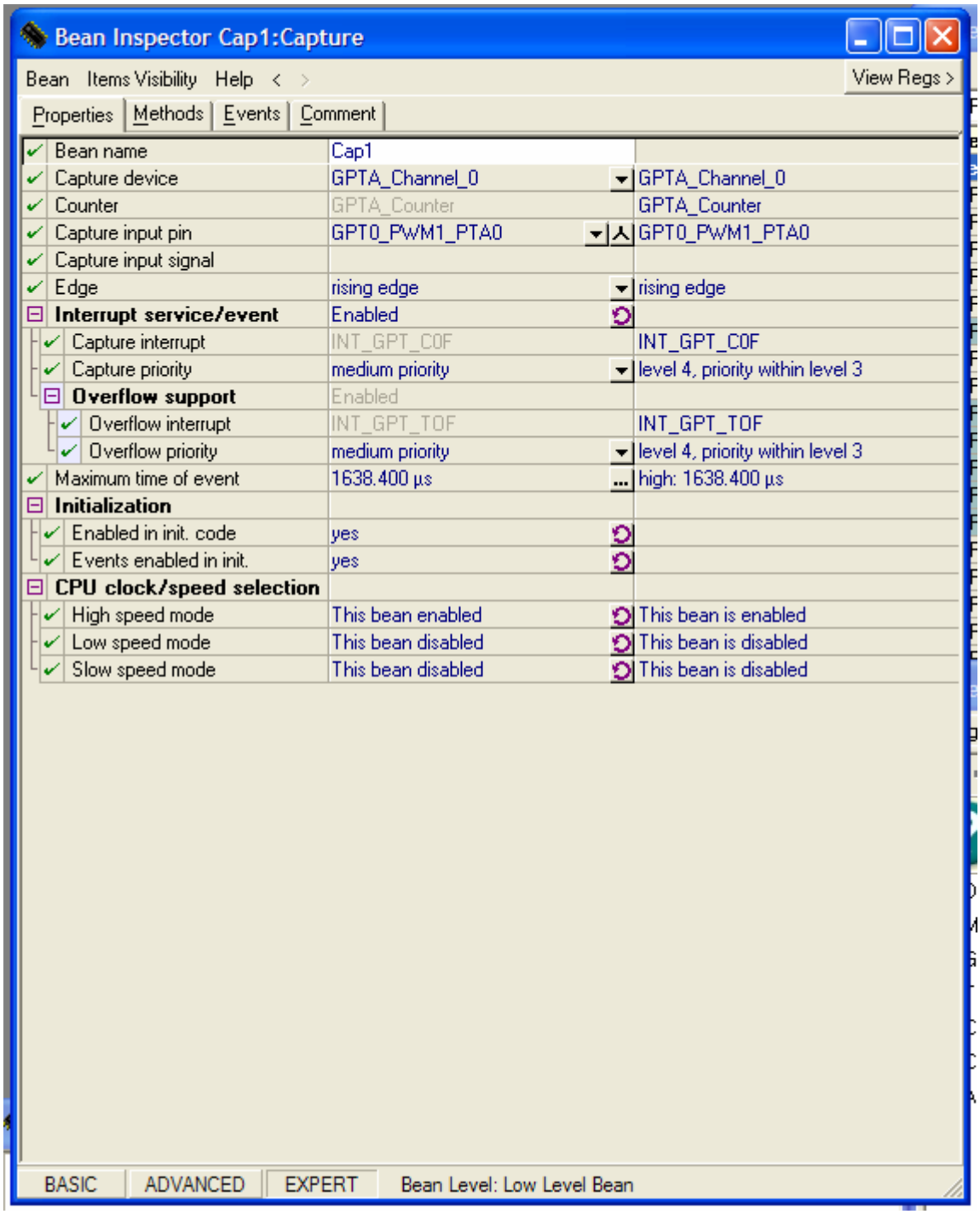

Figura 4.9: Configuració GPT

La configuració és força senzilla. Només hem de tractar dos paràmetres: el flanc d'activació i la freqüència del GPT.

El flanc d'activació està clar que serà el de pujada al iniciar el mòdul, i que l'anirem canviant conforme avancem en l'execució de l'aplicació.

En quant a la freqüència d'execució, optarem per tenir un escalar mínim, el més petit, i compensar-ho a través d'una rutina de tractament de l'*overflow*. Així tindrem una gran precisió però no haurem de tenir els successos pròxims en el temps.
De fet, nosaltres ja sabem que, aproximadament, tindrem un succés cada 5ms (un tren de polsos a 100Hz del que capturem ambdós flancs), i per això volem tenir la màxima precisió possible.
A més, si estem realitzant tractaments de dades que redueixen l'acceleració fins a uns límits de 0'001g, hem de poder capturar aquests valors tant petits (que en temps de "on" són minúsculs) ja que sinó podríem estar afegint errors nosaltres mateixos a través d'una captura poc precisa de les dades.

Havent ja configurat el mòdul, ja només ens queda definir com accedirem a les dades que ens ofereix, és a dir, quin mecanisme d'utilització tenim.

4.2.4. Utilització GPT

Els dos principals mètodes que tenim per accedir a les funcionalitats que ens ofereix el microcontrolador són dues, si utilitzem el Processor Expert:

- Interrupcions del xip
- Funcions del Processor Expert

Interrupcions Associades

Les interrupcions són les rutines que executa el processador quan troba una situació concreta. En aquestes interrupcions hi podem inserir el nostre codi per tal de que tracti allò que nosaltres volem.

Nosaltres utilitzarem dos interrupcions diferents:

- Overflow del registre
- Canvi d'estat del senyal

En la primera interrupció el que farem és definir el tractament que donarem al overflow del registre. En aquest cas el tractament serà mantenir una acumulació dels overflows produïts. Veurem aquesta situació amb més detall al capítol de disseny de l'aplicació.

En el cas de la interrupció produïda per un canvi d'estat del senyal, el que farem és aplicar el procediment que hem detallat pel canvi del flanc d'interrupció i operar amb els valors adequats. També veurem els detalls d'aquestes operacions més endavant en aquesta memòria.

Funcions del Processor Expert

A banda de les interrupcions, també tenim disponibles diverses funcions del PE que ens permeten adquirir dades, canviar configuracions, etc.

De les disponibles per al mòdul GPT, nosaltres només utilitzarem una: la funció GetCaptureValue. Aquesta funció ens retorna el valor del comptador en un moment determinat.

```
byte Cap1_GetCaptureValue(Cap1_TCapturedValue *Value)
```

Codi 1: Funció GetCaptureValue

En concret, si cridem aquesta funció al principi de la rutina de tractament de la interrupció del canvi d'estat, estarem obtenint el valor del comptador en el canvi de l'estat, que ens servirà per a calcular el temps de "on".

4.3. I^2C

El mòdul I^2C ens permet establir comunicacions utilitzant el protocol I^2C. Aquest protocol estableix la comunicació entre diversos dispositius a través de dos canals, un de rellotge i un de dades. És força utilitzat ja que realitza la comunicació a *byte* a *byte* de forma molt senzilla.

4.3.1. Característiques I^2C

El fet que aquest mòdul de I^2C sigui compatible amb la versió 2.1 de la definició I^2C realitzada per Philips, ens garanteix uns mínims de compatibilitat amb altres dispositius. A més, el fabricant ens brinda aquestes característiques:

- Compatibilitat amb dispositius de 3'3v.
- Possibilitat de tenir més d'un *master* al bus.
- Ús del bit de ACK.
- Freqüència configurable.
- Gestió d'interrupcions.
- ID d'esclau.
- Detecció de bus ocupat.

A part d'aquestes característiques, el fabricant ens indica també que el bus de dades pot operar a 100Kbps o bé a la càrrega màxima que es pugui sotmetre dins del límit de capacitància del bus de 400pF.

4.3.2. Funcionament I^2C

El funcionament del mòdul de I^2C és força senzill. L'únic que fa és implementar les diferents funcionalitats que especifica el protocol I^2C en el nostre processador. Podem trobar una adaptació d'aquest protocol al nostre projecte en l'annex C d'aquest document.

L'esquema de registres d'aquest mòdul és facilitat per el fabricant, tot i que sense una explicació més conceptual del protocol, no es pot entendre el seu funcionament. Trobem aquest esquema, però, en la figura 4.10.

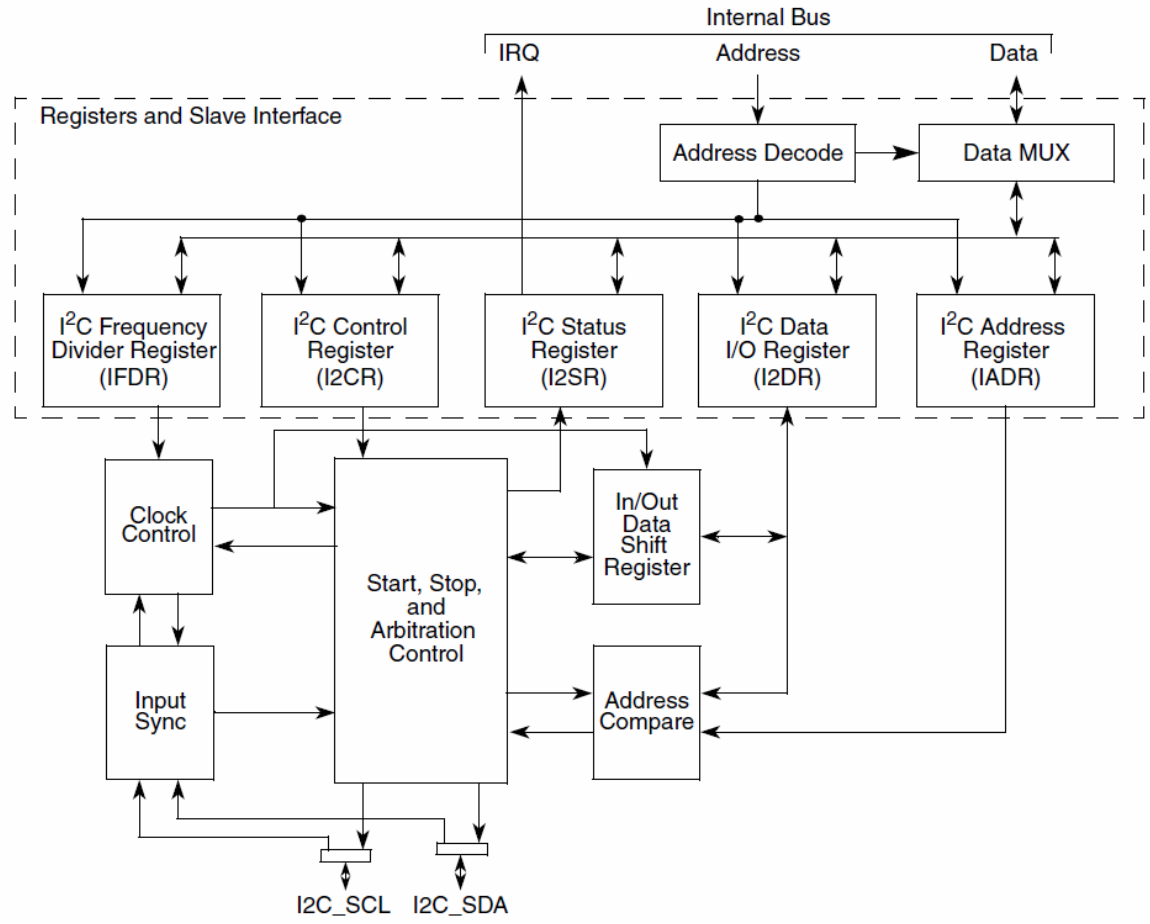

Figura 4.10: Diagrama de Blocs I^2C

Essencialment, el protocol I2C permet transmetre bytes de dades, és a dir, enviar i rebre paquets de 8 bits. Per a fer-ho, es basa en un algorisme força senzill:

1. El *master* s'apodera de l'ús del canal posant-lo a zero.
2. El *master* envia un *byte* amb el identificador del *slave* i un bit que indica si vol escriure en el canal o bé que deixa que sigui el *slave* el que transmeti.
3. El *slave* respon amb un *ack* al paquet que ha rebut del *master*.
4. si el *master* vol escriure en el canal, inicia la transmissió. Si per contra és el *slave* qui ha d'escriure, serà aquest qui comenci a enviar dades.
5. Després de cada paquet de dades enviat el receptor ha de confirmar la recepció correcta d'aquest mitjançant la posada a zero del canal.
6. Un cop el *master* ha acabat la transmissió o la recepció, pot continuar utilitzant el canal amb un altre *slave* o bé pot enviar un STOP per alliberar el canal.

Si considerem que tenim correctament configurat el nostre processador (segons les instruccions del següent apartat) el que podrà succeir és que ens arribi un paquet amb la nostra adreça i una indicació de si el *master* vol llegir o escriure en el nostre processador o bé que vulguem establir-nos nosaltres com a *master* i enviar o rebre dades.

En el primer cas, el que passarà és que saltarà una interrupció del processador que ens indicarà que hem estat activats. Dins d'aquesta interrupció haurem de tractar el que pugui passar: si se'ns vol enviar dades, llegir-les i si es vol llegir dades de nosaltres, enviar-les.

En el segon cas, haurem d'establir-nos com a *master* (activant un bit en concret) i prendre el control del canal. En això el propi circuit hardware del nostre xip ens ajudarà, i ja disposa de la funcionalitat implementada. Aleshores haurem d'enviar l'adreça del xip al que vulguem dirigir-nos i la indicació d'escriptura o lectura. També haurem d'establir les rutines per a aquestes dues operacions.

En el xip, haurem d'interactuar amb els registres adequats per tal de poder enviar i rebre dades. No existeix una llibreria de funcions del Processor Expert per al bus I^2C, per tant, haurem d'elaborar nosaltres mateixos les funcions.

El primer que cal fer és conèixer els registres dels que disposem. Per això consultarem el diagrama de la figura 4.10. Com podem veure, el centre del mòdul es basa en els següents registres:

- I^2C Adress Register (I2ADR): és el registre on s'emmagatzema l'adreça I^2C del node.
- I^2C Frequency Divider Registre (I2FDR): definició de l'escalar que modifica la freqüència del mòdul.
- I^2C Control Register (I2CR): és la configuració del mòdul.
- I^2C Status Register (I2SR): reflexa l'estat actual del mòdul.
- I^2C data I/O Register (I2DR): és on s'emmagatzemen les dades que arriben o bé les que es volen transmetre, és com el *buffer* de dades.

Així doncs, sense interrupcions hauríem d'escriure la configuració del nostre xip al registre I2CR i I2FDR, la nostra adreça d'esclau en el registre I2ADR i finalment consultar el I2SR per saber quan hem de llegir dades o escriure'n al I2DR. Amb les interrupcions i el CodeWarrior (CW) això és força més senzill.

Deixarem per l'apartat de configuració la manipulació dels registres I2CR i I2FDR, mentre que en l'apartat d'utilització definirem com accedir a les funcionalitats del mòdul amb CW i Processor Expert.

4.3.3. Configuració I^2C

De la mateixa forma que en el cas del mòdul ADC, per a la configuració del mòdul de comunicació I^2C ens basarem en l'aplicació de CodeWarrior que ens permet generar el codi de configuració directament.

Els paràmetres a configurar són els següents:

- Freqüència del mòdul
- Mode Master/Slave
- Transmit/Recieve
- Enviament de ACK
- ID Slave
- Pins de rellotge i dades
- Interrupcions

En el nostre cas, intentarem treballar amb la màxima freqüència disponible, ja que al tractar-se d'un bus general del sistema, es preveu que hi hagi més sensors connectats, i per tant la velocitat de transmissió ha de ser elevada.

La decisió de si iniciar el mòdul com a Master o bé Slave ve donada pel protocol propi que s'ha definit en l'annex C d'aquest document. Els sensors associats al sistema seran tots esclaus d'un únic Master.

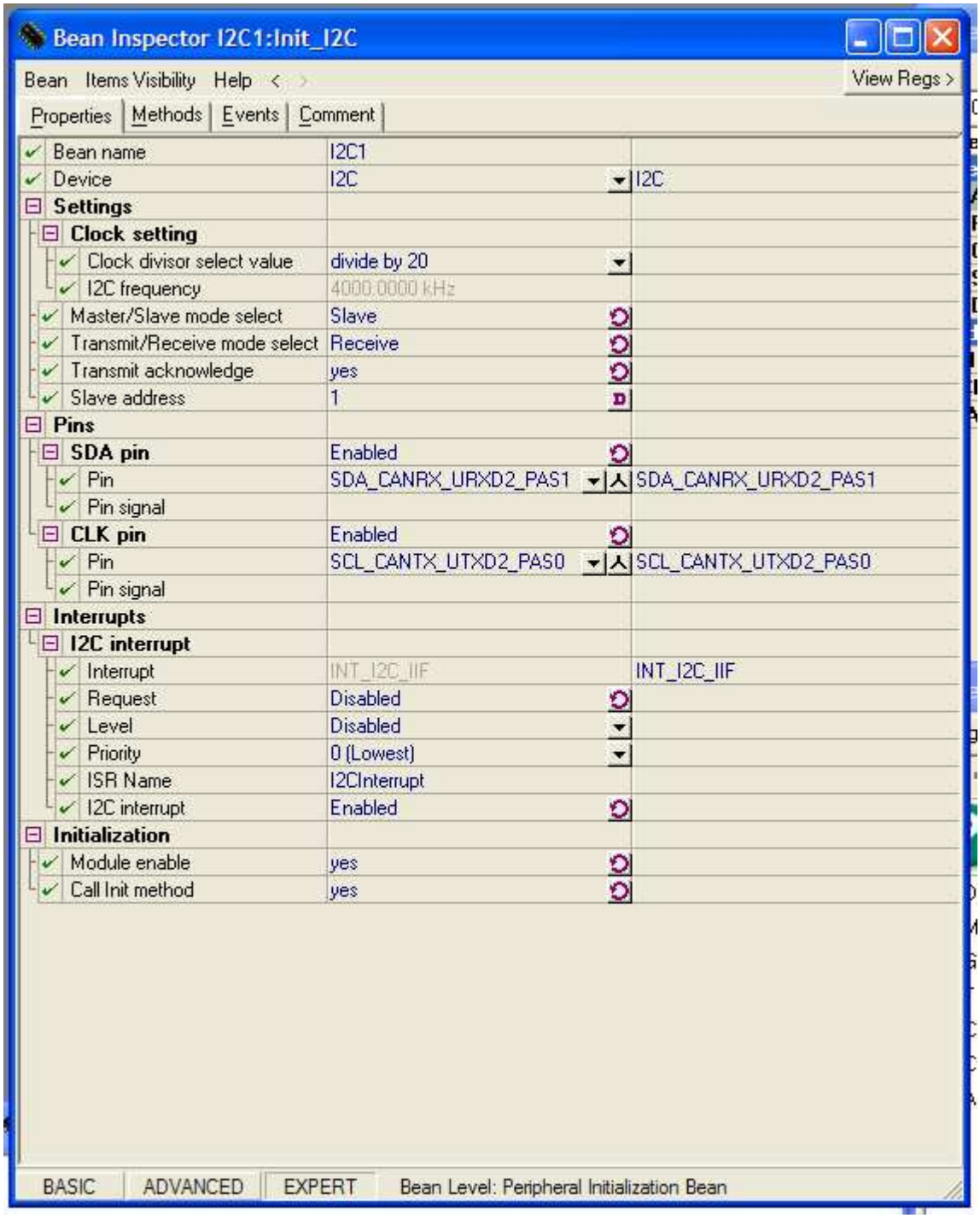

Figura 4.11: Configuració I^2C

Segons el mateix protocol propi, aquests sensors esperaran una configuració del gestor del bus, per tant, estarà al mode recieve.

Activarem l'ús de senyals d'ACK per a indicar al Master que hem rebut correctament les dades enviades. Serà aquest gestor del bus qui ens assignarà prèviament a l'inici del sistema un identificador únic.

Finalment, definirem com a pins de dades i rellotge els que ens indica el fabricant.
Podem trobar aquesta configuració en la figura 4.11.

L'últim que hem de fer és definir el nom que hem assignat a la interrupció que tractarà els successos relacionats amb aquest mòdul, això és, al rebre el ACK d'un *byte*, al rebre un paquet amb la nostra adreça o bé al perdre el control sobre el bus. Veurem un ús d'aquestes situacions en el següent apartat.

4.3.4. Utilització I^2C

Un cop realitzada la configuració inicial, tot ho farem a través de la interrupció que gestiona el mòdul. Trobarem dos possibles situacions: una en la que rebem una configuració a aplicar, i l'altra en la que hem d'enviar les dades de l'acceleració.
Ambdós casos es troben iniciats per la recepció d'un paquet amb la nostra adreça des del gestor del bus, per tant, els dos casos es podran gestionar des de la interrupció.

Hem de recordar que aquesta interrupció haurà d'interactuar amb els registres vistos en aquesta explicació a baix nivell. Veurem com fem aquesta aplicació en l'apartat corresponent al disseny de l'aplicació.

5. CodeWarrior IDE: l'Entorn de Desenvolupament

El fabricant Freescale (com a part de Motorola) ens ofereix un entorn de programació professional dedicat als xips que fabrica. Disposa d'una versió general que adapta a les diferents famílies de processadors amb els que tracta. Aquest entorn s'anomena CodeWarrior.

Concretament nosaltres disposarem d'una versió de CodeWarrior específicament adaptada per a la família de processadors ColdFire. Aquesta edició disposa de tot el necessari per a crear una aplicació enfocada a aquesta arquitectura.

Figura 5.1: Informació de CW

Una dels principals avantatges d'utilitzar un entorn de programació consolidat com és CodeWarrior, és el fet que es troba disponible per a diverses plataformes. No estem parlant ara de les

diverses arquitectures i famílies de xips a les que està enfocat, sinó sobre els múltiples sistemes operatius sobre els que es pot executar. Disposa, segons el fabricant, de suport per a Windows, Mac OS, Linux i Solaris. A més a més, defensa que manté una aparença pràcticament igual en tots els sistemes, per tal de fer més senzill el pas d'un a un altre.

A banda del fet que es tracta d'un entorn multi-plataforma, també cal valorar els llenguatges amb els que ens permet treballar, ja que un entorn que podem executar en molts sistemes, però que només ens permet treballar amb un llenguatge de baix nivell, no és un bon sistema.
Concretament, CW treballa amb 3 llenguatges d'alt nivell: C, C++ i Java. Sempre segons l'arquitectura a la que està enfocat. En el cas de la família ColdFire, aquestes opcions es redueixen a C i C++.
Tampoc podem oblidar que en totes les famílies permet la inclusió de codi en llenguatges de baix nivell.

Tot i que moltes vegades els programadors que no estan acostumats a treballar amb sistemes embedits no hi paren atenció, una de les grans virtuts dels entorns de programació integrats és el fet que et permeten escriure en el xip amb només un parell de clics. És el cas del CW que ens ofereix una eina per esborrar i escriure el nostre MCF5213 sense cap complicació. A més, disposem d'una funció que integra la compilació, l'enllaçament i l'escriptura en el xip.

Aquesta independència entre el codi i l'arquitectura a la que va dirigit, ens aporta un benefici en quant a consistència. La consistència en aquest cas s'entén com la separació que tenim entre implementació i hardware sobre el que s'ha implementat.

En certa manera, aquesta forma de treballar ens afegeix una capa d'abstracció del hardware que hi ha radere; nosaltres només hem de tractar amb el CodeWarrior i no amb el ColdFire.

Finalment, cal remarcar també la possibilitat d'ampliació que ens ofereix l'entorn a través de plug-in, és a dir, a través d'afegir petites aplicacions que ens aportin noves funcionalitats o millorin les prestacions de funcionalitats ja existents en la plataforma.

A continuació, en l'apartat 5.1, veurem el cicle de programació que es segueix en el entorn CodeWarrior. Més endavant analitzarem els components detallats de l'entorn i acabarem aquesta secció amb una explicació sobre els diferents nivells d'abstracció que ens ofereix CW.

5.1. Cicle de Programació

El propi fabricant ens indica el cicle de programació que es segueix amb el seu entorn. El fabricant proposa un cicle de programació exclusivament orientat a l'ús del seu entorn, per tant, no considera en aquesta descripció fases tant importants com el disseny de l'arquitectura o l'anàlisi de requeriments:

- Tenir una idea per a un nou software
- Implementar la idea en codi
- Compilar el codi
- Enllaçar el codi
- Corregir els errors
- Produir una versió final del producte

No podem confondre aquest cicle de programació amb el cicle de desenvolupament de software. El que explicarem en aquest apartat és únicament aquelles parts del procés de desenvolupament de software que estan relacionats amb l'eina CodeWarrior.
Per a no confondre al lector indicant que aquesta seqüència de passos pot representar una forma vàlida de desenvolupament de software hem optat per no numerar-los per a no indicar un ordre o precedència.
Explicarem els passos del procés de desenvolupament de software en l'aparta corresponent.

Ens centrarem doncs, en els passos principals que ens permet realitzar el CW, és a dir, des d'implementar el codi fins a produir una versió definitiva del software. Podem relacionar aquestes activitats a través d'un diagrama (figura 5.2) per a veure com proposa el fabricant que s'han de realitzar.

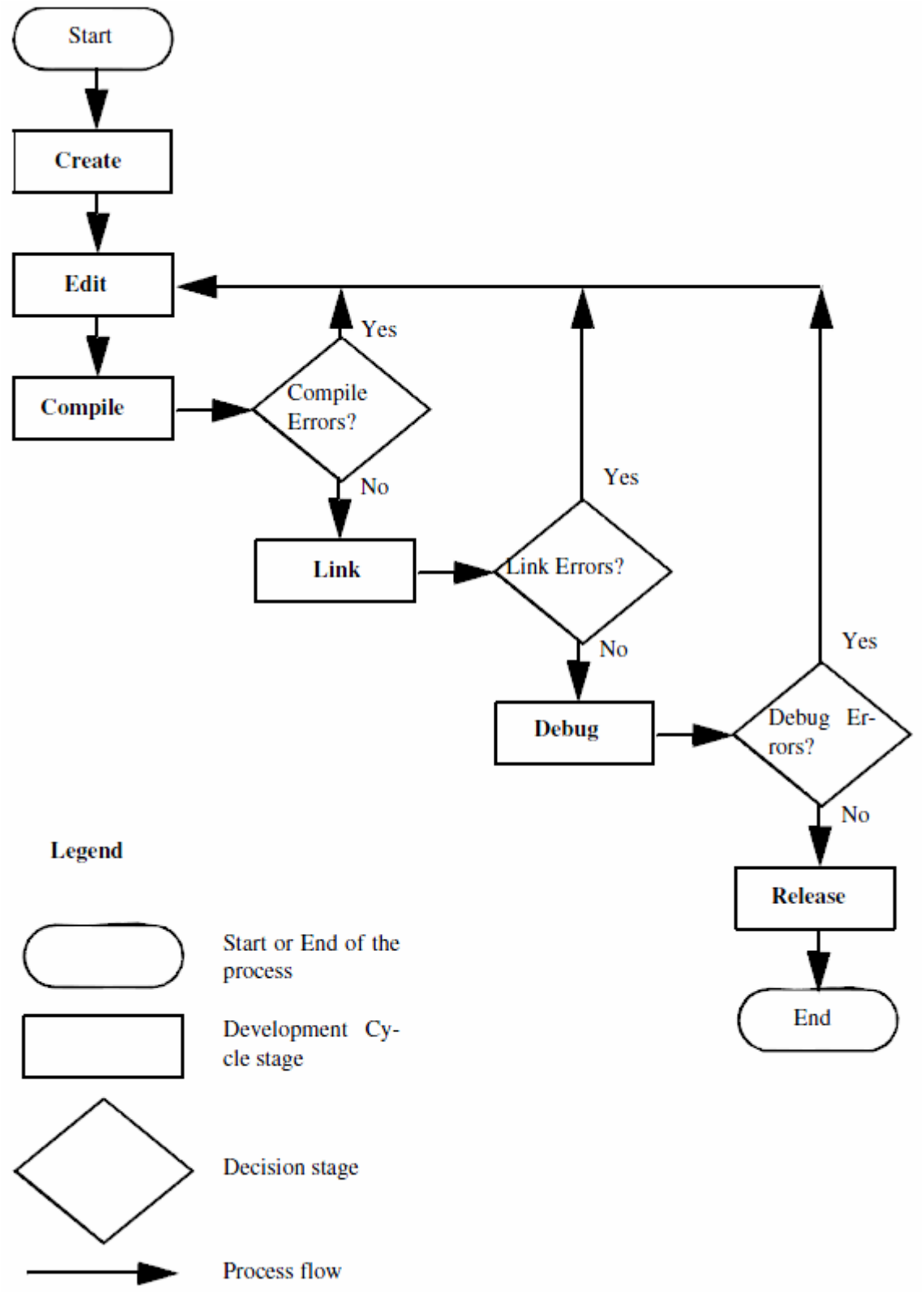

Figura 5.2: Cicle de Programació

Definirem ara les funcionalitats que ens permet desplegar cada una de les activitats proposades:

- **Crear** (*create*): es tracta d'inicialitzar el projecte, amb la configuració específica del xip i el codi software que ens permet aplicar la configuració general al MCF5213.
- **Editar** (*edit*): afegir el codi propi a l'estructura del projecte creada per CW. Aquí és on farem que l'aplicació faci el que nosaltres volem. Podrem crear funcions, interfícies, classes, rutines, i tot el que hem vist en llenguatges d'alt nivell.
- **Compilar** (*compile*): es tracta de convertir el codi d'alt nivell en un llenguatge màquina que s'executi en el nostre xip. Transformarà instrucció a instrucció, així com afegirà les interrupcions necessàries a la taula, etc.
- **Enllaçar** (*link*): generarem un únic fitxer binari executable per el host destí. Aquest serà el fitxer que copiarem a la memòria Flash del dispositiu.
- **Depurar** (*debug*): és la part en la que podrem observar pas a pas l'execució del nostre codi, així com els valors de les variables, les crides a funcions, etc. Fent ús d'aquesta eina podrem saber on fallen les nostres aplicacions i corregir aquests errors.
- **Produir versió final** (*release*): quan haguem acabat el desenvolupament de l'aplicació, farem una versió final del producte ja empaquetada i llesta per a copiar al dispositiu i executar.

Com veurem a continuació, tenim diversos components de la IDE CodeWarrior que ens faciliten les tasques que abans hem citat. El següent apartat intenta ser una explicació d'aquestes eines.

5.2. Components del CodeWarrior

L'entorn de desenvolupament CodeWarrior ens ofereix diverses eines per a realitzar tot allò que vulguem amb el nostre producte software. Són eines que ens ajuden des de l'organització fins a la realització de proves.

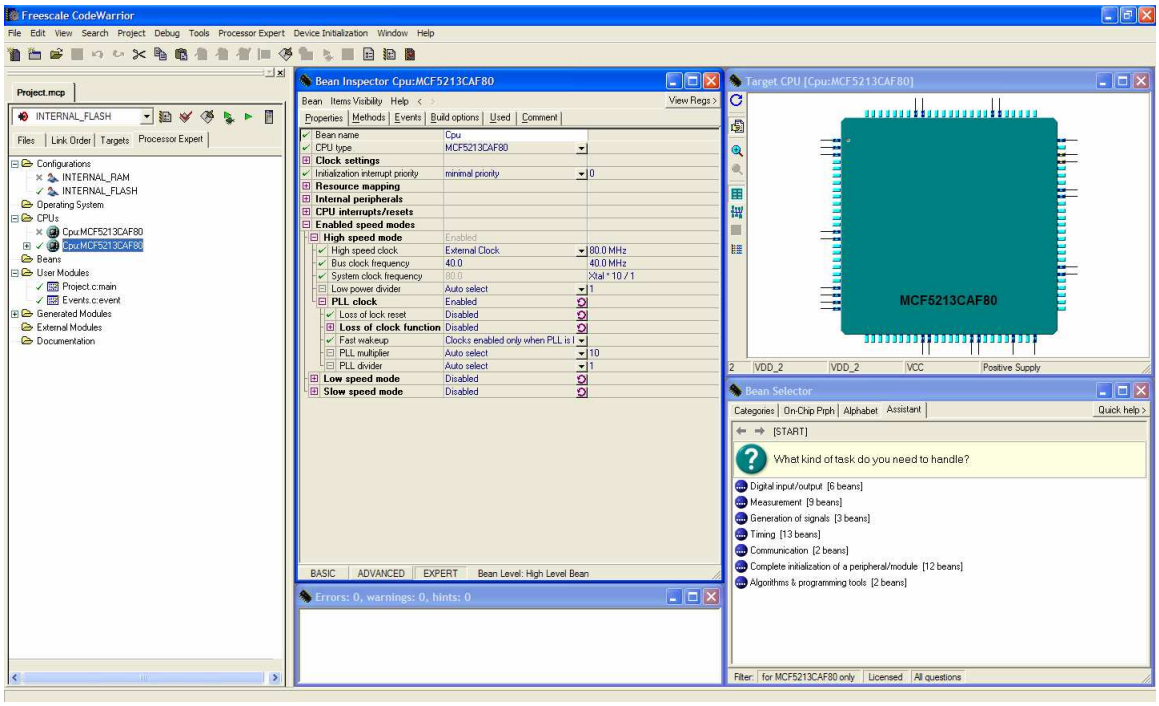

Figura 5.3: Entorn CW

Ja en la pantalla principal de l'aplicació (figura 5.3), podem distingir diverses d'aquestes eines. A la banda esquerra trobem el gestor de projectes i a la part superior trobem els accessos al sistema de construcció i depuració de les aplicacions.

En concret, les eines que ens ofereix l'entorn (ja sigui a través d'accessos o bé directament sobre el IDE) són les següents:

- Gestor de Projectes
- Editor
- Motor de Cerca
- Navegador de Codi

- Sistema de Construcció
- Depurador
- Programador del Xip

Els veurem amb més detall a continuació.

Gestor de Projectes

El Gestor de Projectes és una eina que s'encarrega de controlar tot el relacionat amb els fitxers del projecte. Portarà un control sobre quins fitxers s'han modificat des de l'última construcció del sistema, quins cal tornar a compilar, etc.

Una de les funcions claus d'aquesta eina és el control que porta sobre l'ordre de compilació i enllaçament dels diversos fitxers a l'hora de construir el sistema, ja sigui per a fer proves o per a una lliurament final.

S'encarrega també del control dels objectius, és a dir, de definir per a quin processador estem desenvolupant el nostre projecte i sobre quina de les memòries del xip (RAM o flash) escriurem les nostres instruccions.

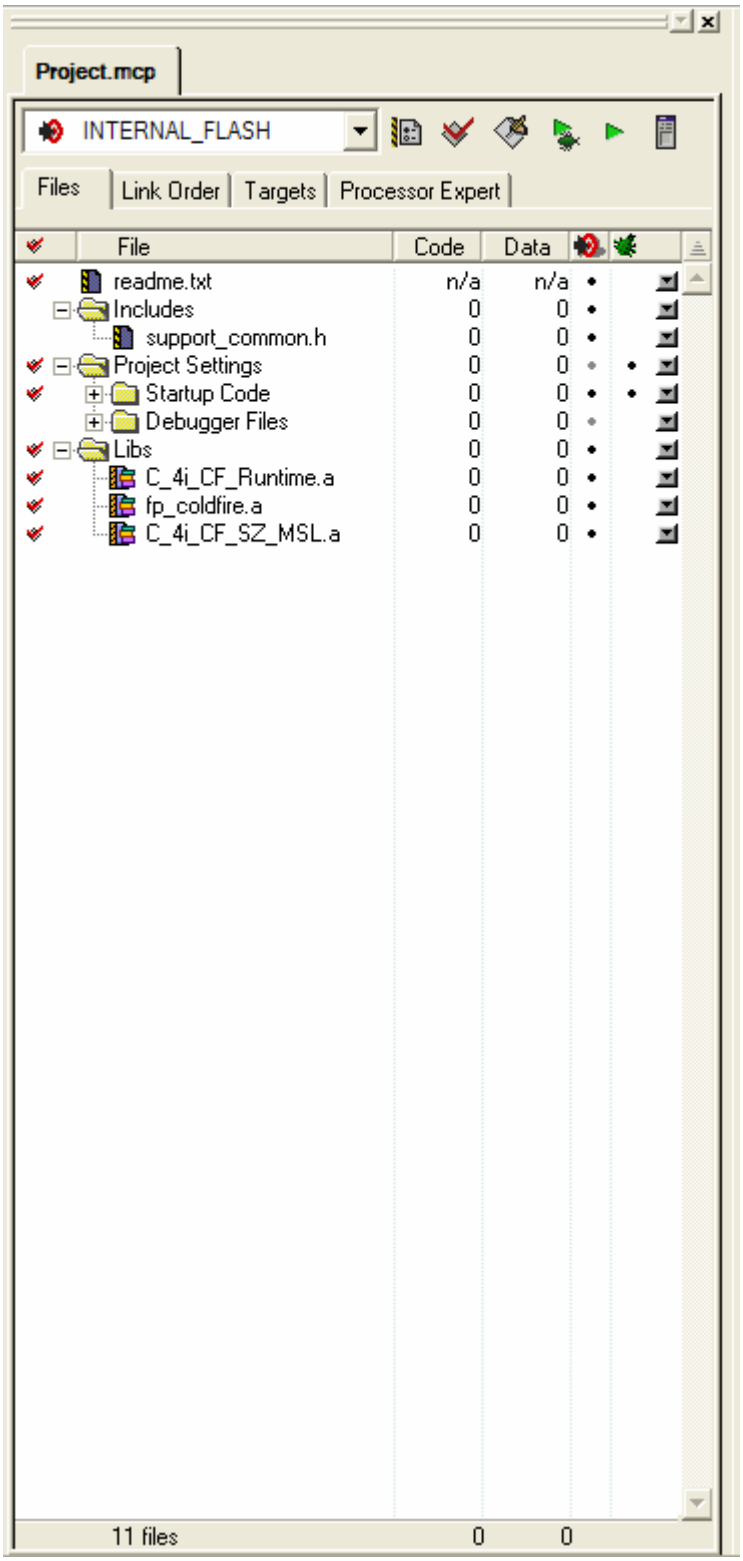

Figura 5.4: Gestor de Projectes

Trobem aquesta eina sempre activa en el panell esquerra de la pantalla principal de l'aplicació (figura 5.4). Hi podem trobar una casella que ens permet escollir la memòria destí del projecte, uns

botons amb diversos accessos a altres eines i unes pestanyes amb diverses funcions:

- Navegador d'Arxius
- Ordre d'Enllaçament
- Memòries Objectiu (RAM o flash)
- Processor Expert

On l'última (PE) només estarà activa per aquells projectes que en facin ús.

Editor

Aquest editor és l'eina principal per a escriure el nostre codi. Ens ressaltarà el codi per a diferenciar les variables, de les funcions, de les constants, etc. També ens comprovarà la correcta sintaxi de parèntesis i claus, a més de permetre'ns navegar entre les funcions.

Aquesta última funcionalitat és molt útil per a conèixer la implementació de les funcions que ens ofereix el propi entorn a través del Processor Expert (que tractarem més endavant).

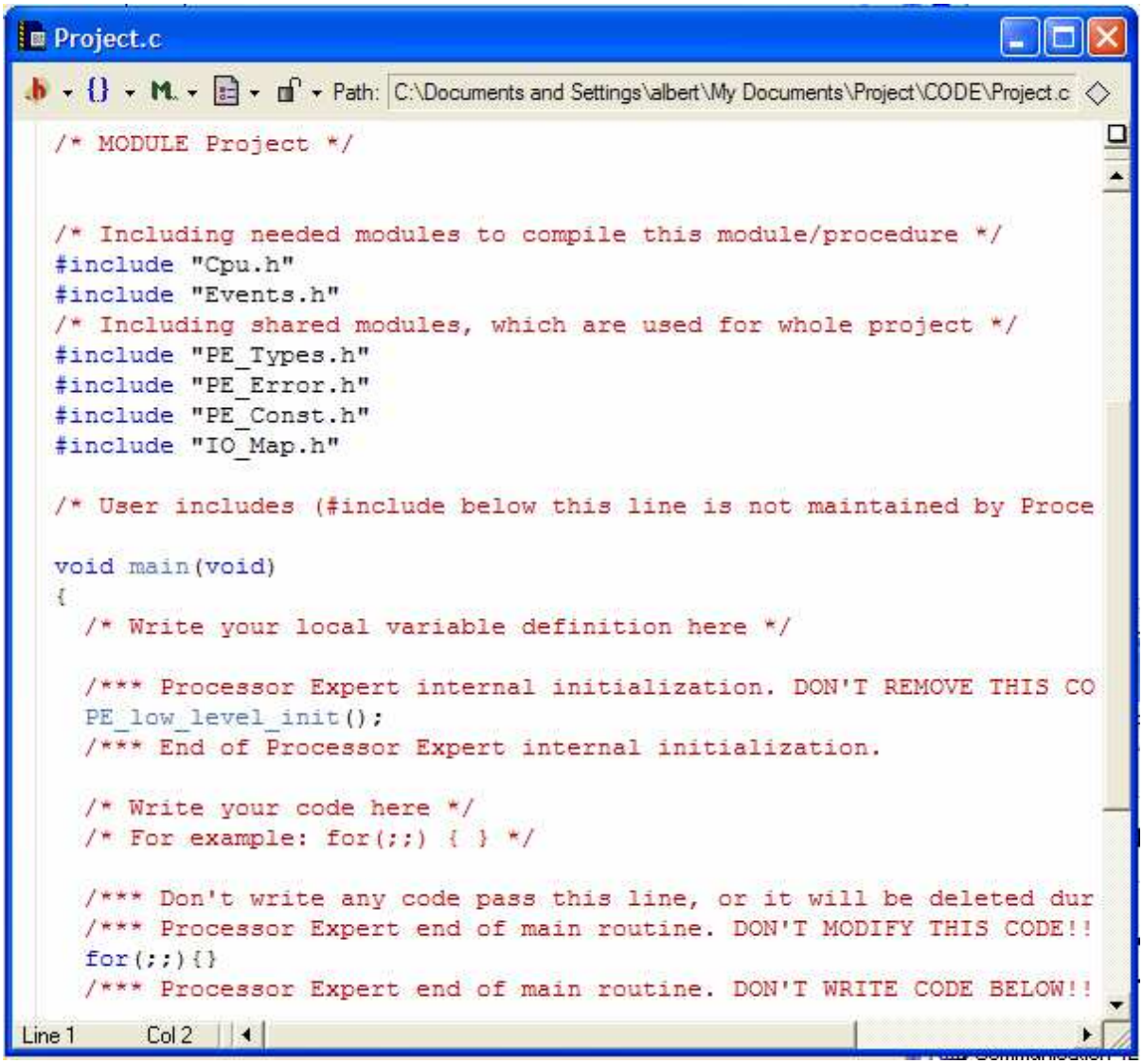

Figura 5.5: Editor del CW

Motor de Cerca

Si el que volem és trobar una variable o una expressió en el nostre codi, aquesta eina ens facilitarà la tasca. Ens permet realitzar cerques tant per coincidència exacta com utilitzant expressions regulars que ens defineixin sentències diverses. Un cop hem trobat allò que busquem, podem reemplaçar-ho des de la mateixa eina, i aplicar els canvis a les diverses ocurrències de l'expressió.

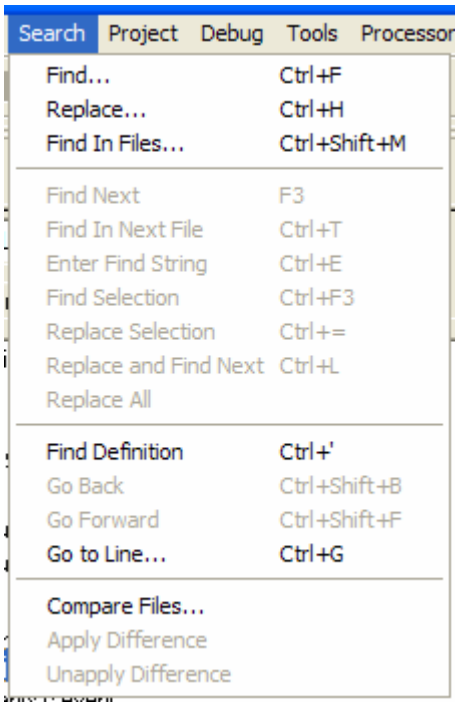

Figura 5.6: Panell de Cerca

Ofereix també una opció per a trobar definicions concretes de variables o funcions, per tal de poder navegar fàcilment i trobar l'històric d'una variable amb facilitat.

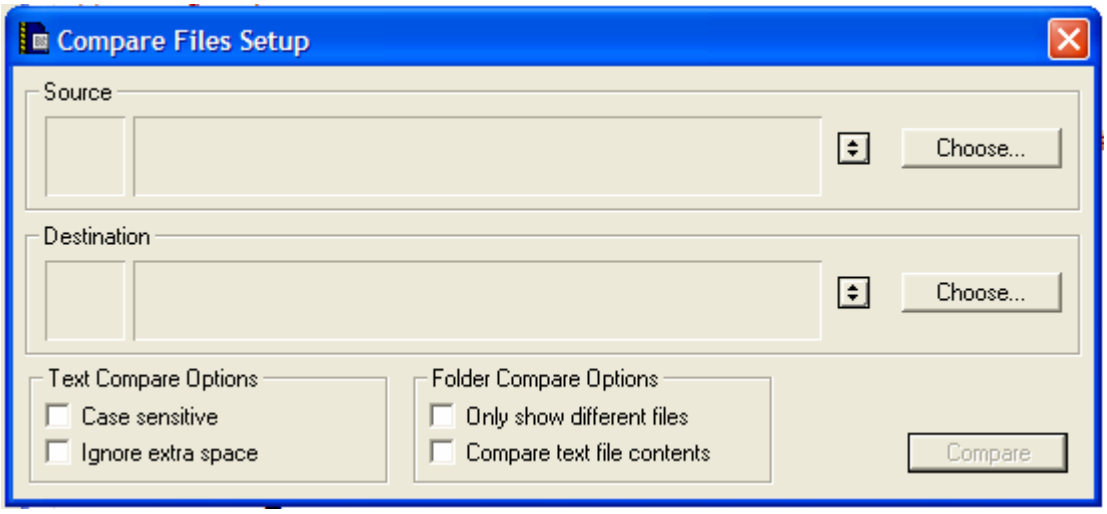

Figura 5.7: Finestra de Comparació

Finalment, ofereix una funció de comparació d'arxius en busca de diferències. Aquesta funció resulta vital per a tractar amb diferents versions d'un mateix projecte.

Navegador de Codi

El navegador de codi manté una base de dades de tots els objectes i símbols del codi. Manté així un entramat de relacions entre tots els objectes del projecte: de les instàncies amb les seves declaracions, de les declaracions amb els usos, de les funcions amb les seves crides, de les crides amb les implementacions, etc.

Aquest sistema es troba desenvolupat tant per llenguatges procedurals com per a llenguatges orientats a objectes.

Sistema de Construcció

S'encarrega de compilar el codi necessari i de enllaçar-lo. Seguirà l'ordre que li indica el gestor de projectes, ja sigui el que apareix per defecte o el que hagi modificat el programador a través de la pestanya corresponent del gestor.

Ens ofereix la possibilitat d'utilitzar components diferents dels que el propi entorn ens instal·la per defecte, convertint-lo així en un sistema altament modulable i tolerant a canvis i preferències de l'usuari.

Tenim a la nostra disposició un botó  que respon a la llegenda "Make". Aquest botó el que farà és primer compilar i després enllaçar l'aplicació. També disposem d'un botó  que directament llançarà el "Make" i ens executarà l'aplicació en el processador.

Depurador

El depurador (*debugger* en anglès) és l'eina que ens permet detectar errors en el nostre codi. La clau principal del seu funcionament és l'execució individual de sentències de codi, és a dir, poder executar les instruccions una a una, sense haver d'esperar al final de l'execució per a veure el valor de les variables.

Per al control més avançat de l'execució podem definir punts de parada (o *breakpoints*), és a dir, punts en que el depurador pararà l'execució del programa per a que puguem veure l'estat actual del procés. Indicarem aquests punts de parada en el propi codi, marcant la fila en concret amb un punt vermell.

Gràcies al seu analitzador de variables, podem veure en cada instant el valor de totes les variables actives en l'execució d'una rutina. A més a més, podem escollir el format en el que volem veure els valors, des de enters, fins a hexadecimal, passant per binari o caràcter.

En la finestra de treball del depurador, podem veure tres zones diferenciades. A la part superior esquerra trobem una pila de les funcions que s'han cridat i a la part dreta trobem el visualitzador de les variables actives en la última funció de la pila.

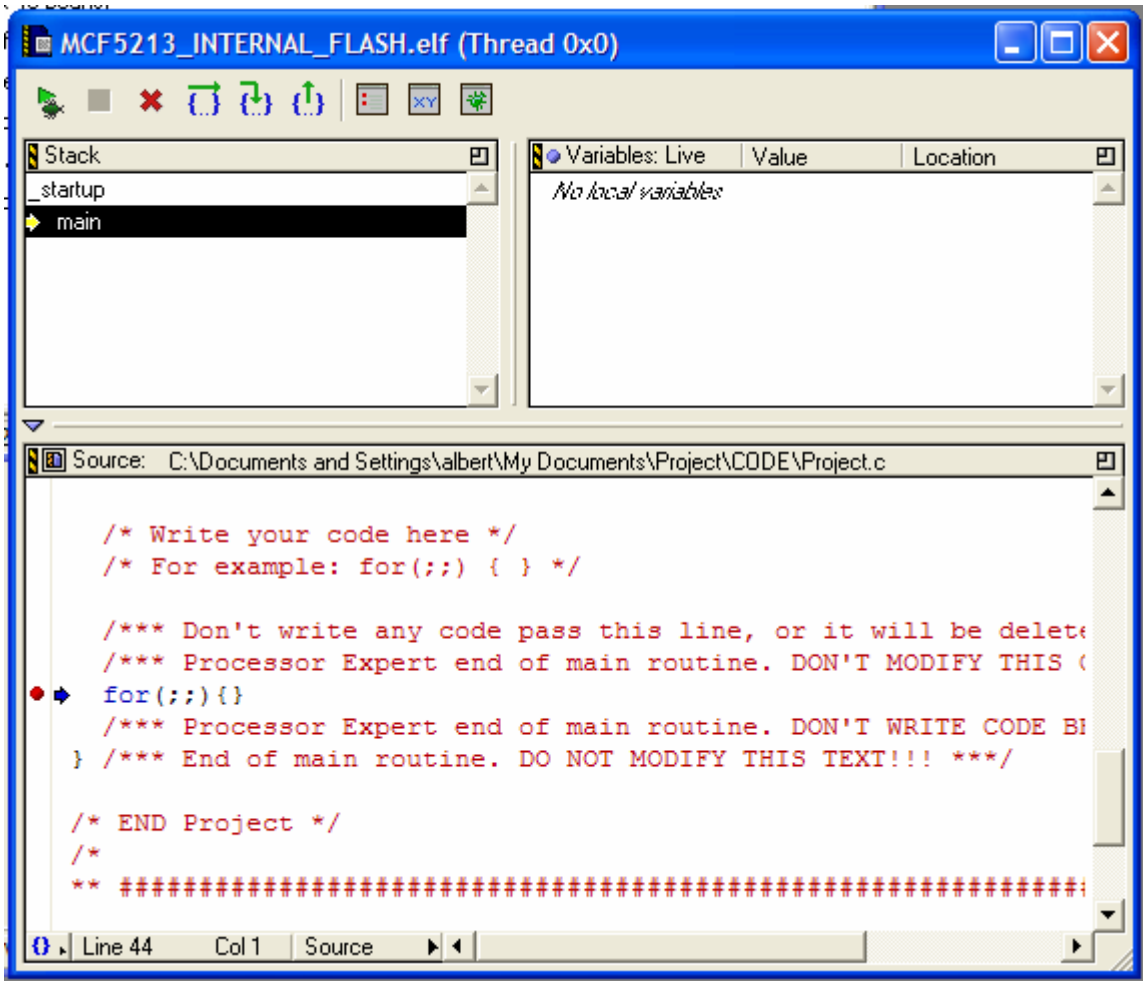

Figura 5.8: Finestra Depurador

Finalment, en la part inferior trobem el codi que s'està executant. En la mateixa finestra podrem decidir què fer, si avançar instrucció a instrucció, entrar en una funció, sortir-ne, avançar fins al següent punt de parada o bé continuar fins al final de l'execució.

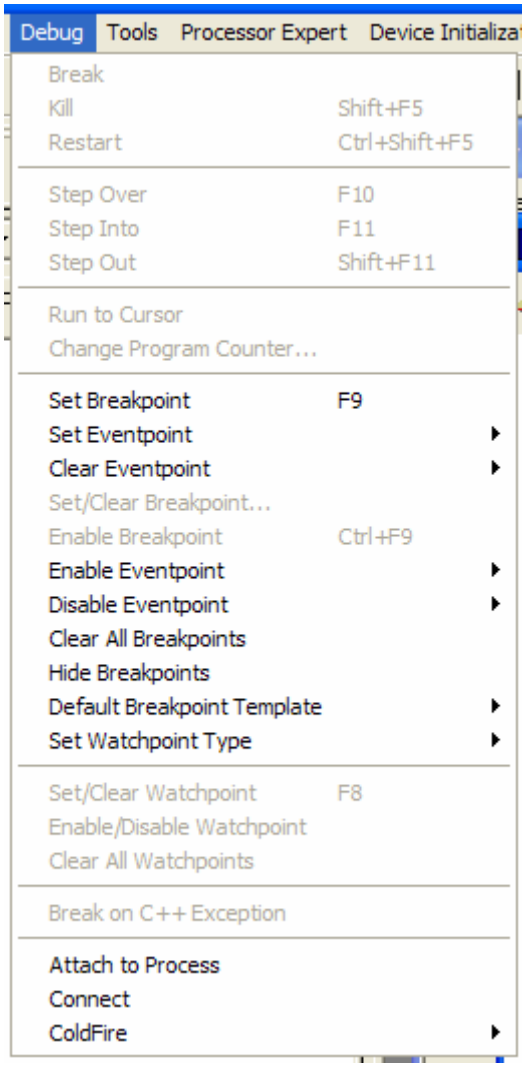

Figura 5.9: Menú Depurador

També podem accedir a diverses funcionalitats de l'eina de depuració utilitzant el seu menú corresponent (figura 5.9). A través d'aquest menú podrem marcar línies de codi per a aturar l'execució, navegar pel codi, establir events, etc.

Programador Xip

Un cop tenim una aplicació construïda (compilada i enllaçada) és el moment de passar-la al processador per tal de provar-la o bé depurar-la.

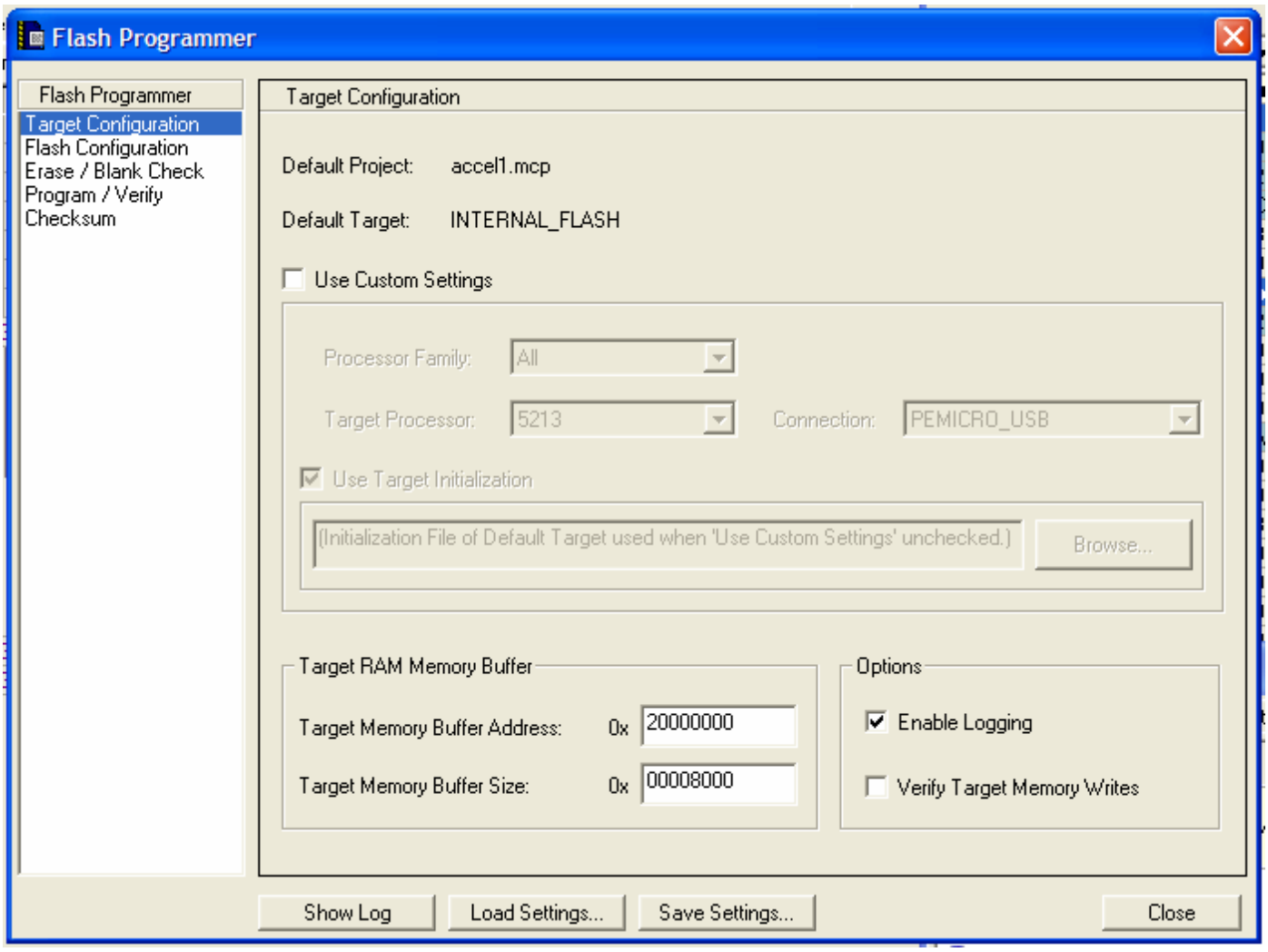

Figura 5.10: Programador Xip

L'entorn CW ens ofereix una eina per a manipular la memòria del nostre xip, ja sigui per esborrar-la, comprovar el seu funcionament o bé escriure-la.

Amb una senzilla finestra (figura 5.10), ens permet indicar quin processador és el que estem programant i les dades sobre la memòria (adreça d'inici, adreça de fi, etc.).

A través de les pestanyes del costat accedirem a la totalitat de funcionalitats que ens ofereix:

- Esborrar la memòria.
- Comprovar que la memòria està en blanc.
- Escriure un programa.
- Comprovar que un programa ha estat escrit.
- Realitzar una comprovació d'integritat.

Un cop seleccionat què volem fer, l'eina s'encarregarà de la resta (establir comunicació USB, codi de connexió, etc.).

Val a dir que aquesta és l'eina que empren els altres components de l'entorn per a comunicar-se amb el xip, des del depurador fins a la funció que ens permet executar directament l'aplicació al xip.

5.3. Rapid Application Developement (RAD)

Fins ara, les característiques que hem definit sobre l'entorn de programació CodeWarrior són més o menys les que ens hauria d'oferir qualsevol entorn. El punt fort d'aquesta *suite*, en canvi, és el que analitzarem en aquest últim apartat.

Desenvolupar aplicacions per a xips implica invertir molt temps en configurar els diferents mòduls utilitzats. Aquesta configuració es fa a través de la manipulació dels registres del processador, feina que resulta molt feixuga i pot acabar en moltes ocasions en un malfuncionament de l'equip.

Però no només per a configurar els mòduls es fa necessari treballar amb registres. Per qualsevol interacció amb el mòdul (ja sigui de configuració o tractament de dades) implica treballar a baix nivell. Per tant, si volem llegir dades d'un mòdul o bé enviar dades, o capturar un temps, haurem de llegir de o escriure en registres.

Per evitar-nos aquesta tasca, en la mesura del possible, i sempre segons les nostres preferències, l'entorn CodeWarrior ens ofereix el que anomena Rapid Application Development (RAD).

RAD està format per dos modes, a escollir : una ens permetrà realitzar la configuració de forma interactiva, sense haver de conèixer els registres, i l'altra ens permetrà realitzar la configuració de la mateixa forma i a més a més ens oferirà una llibreria per interactuar amb cada mòdul.

Així, finalment tenim 3 possibilitats a l'hora de desenvolupar la nostra aplicació:

- Sense RAD
- RAD: Inicialització del Dispositiu
- RAD: Processor Expert (PE)

L'assistent que ens ajuda a crear un nou projecte ens preguntarà, en un dels passos, amb quin mode volem desenvolupar la nostra aplicació (figura 5.11).

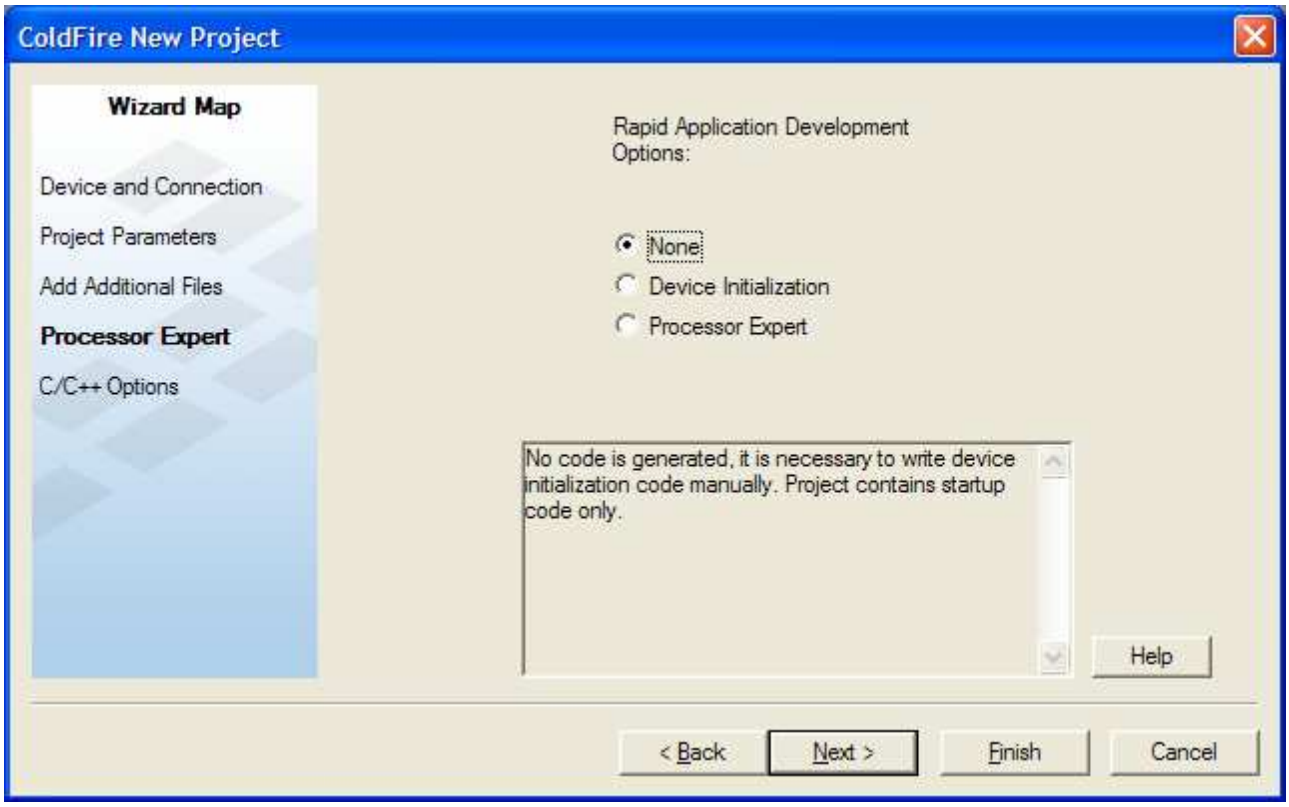

Figura 5.11: Tria del RAD

A continuació exposarem les característiques de cada mode. Anirem de menor a major complexitat del mode, però que en realitat és de més complexitat a l'hora de desenvolupar aplicacions a menys, ja que contra més coses controla el CodeWarrior, menys hem de controlar nosaltres, i més independència del xip obtenim.

5.3.1. Sense RAD

Aquest mode és el que deixa més feina a fer al programador. En aquest cas, el CodeWarrior només ens presenta una arquitectura de projecte on no trobem res. Tota la configuració del dispositiu s'ha de realitzar a mà.

El dispositiu ja ve configurat de tal manera que arranca correctament, però no permet utilitzar cap mòdul exceptuant el d'execució de codi. És al principi d'aquesta execució on s'han d'escriure tots els registres de configuració dels mòduls que vulguem anar arrancant.

No és recomanable treballar amb aquest mode, ja que complica molt la feina del programador, i són moltes les línies de codi per a configurar el dispositiu.

Per a que ens fem una idea, si arranquem el Processor Expert, no afegim cap mòdul i li demanem que ens generi el codi per a l'execució d'una aplicació buida, és a dir, sense cap línia de codi nostra, el Processor Expert generarà 1419 línies de codi.
Aquestes línies són de codi exclusivament de control i configuració. Són les línies que configuren la CPU, les taules d'interrupcions, etc. Però no són línies de codi útils que duguin a terme una funció per al programador. Si emprem el mode sense assistència de RAD, haurem d'escriure tantes línies de codi com necessitem per a suplir aquestes 1419. Segurament serà un nombre menor de 1419 ja que el PE crea funcions que després no utilitzarem, però tot i això seran moltes.

5.3.2. RAD: Inicialització del Dispositiu

El mode d'inicialització del dispositiu ens genera automàticament les instruccions per a configurar els mòduls que vulguem segons els paràmetres que nosaltres sol·licitem.

En arrancar un projecte amb aquesta opció activa apareix una finestra que ens permet activar mòduls (figura 5.12). Podem seleccionar en aquesta finestra els mòduls que volem activar. Al fer-ho ens apareixerà una segona finestra que ens permetrà configurar els valors del mòdul en concret (figura 5.13).

En aquesta finestra ens apareixen tots els camps configurables (a l'esquerra) on podrem posar els valors que vulguem. En el cas de tenir diversos valors disponibles, en els indicarà i ens donarà explicacions (a través de finestres emergents) sobre quin significat té cada camp.
A la dreta trobarem els registres relacionats, i podrem anar veient com canvien els valors d'aquests registres quan realitzem canvis en les opcions.

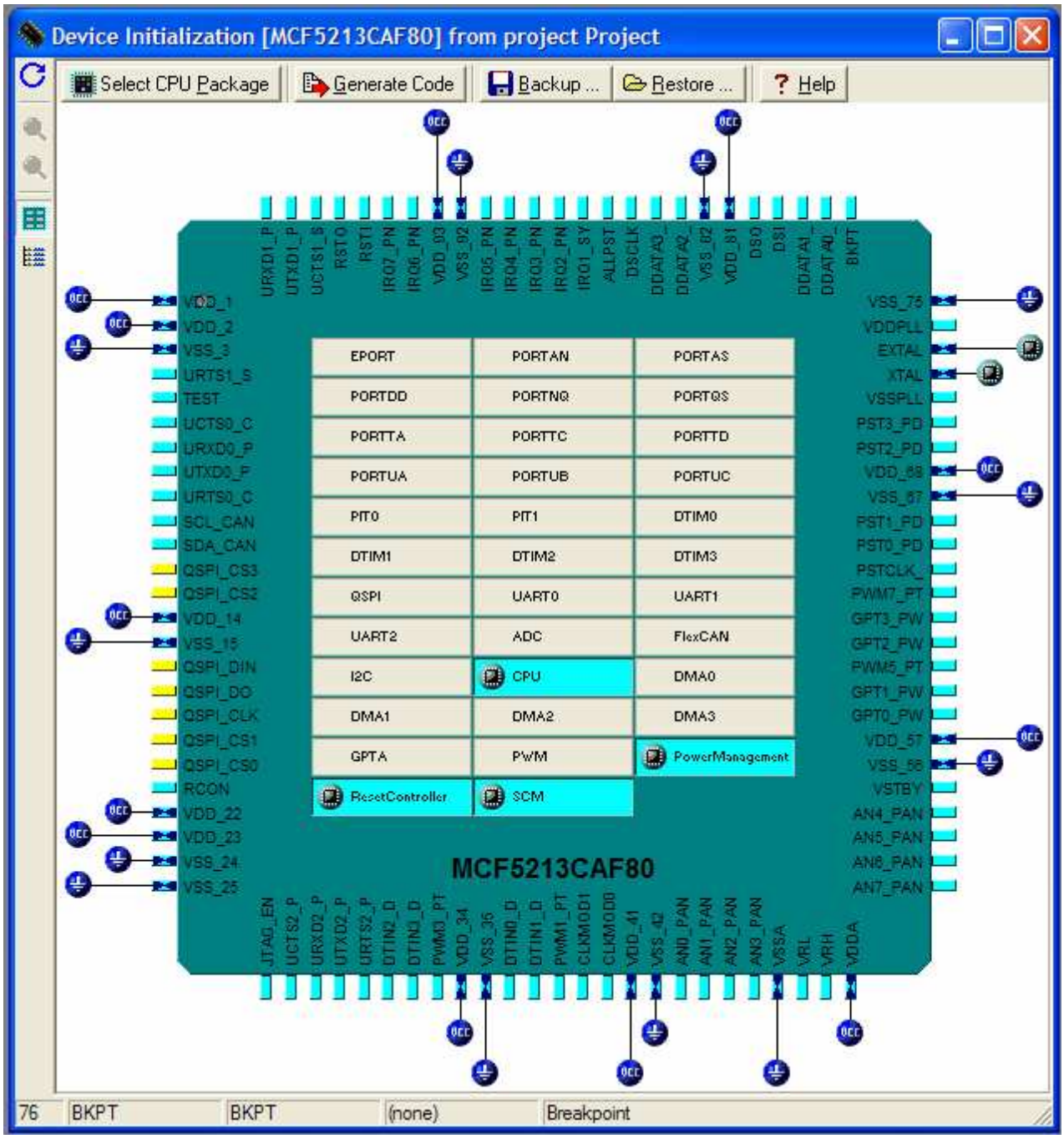

Figura 5.12: Mòduls Inicialitzables

D'aquesta manera, al només canviar els registres a través del formulari que CodeWarrior ens ofereix, ens assegurem que no configurarem cap registre de forma equivocada o contradictòria.
Al acabar la configuració de tots els mòduls, podrem generar el codi associat a aquestes inicialitzacions. Aquest codi serà el que s'inclourà en el projecte i s'executarà a l'inici.

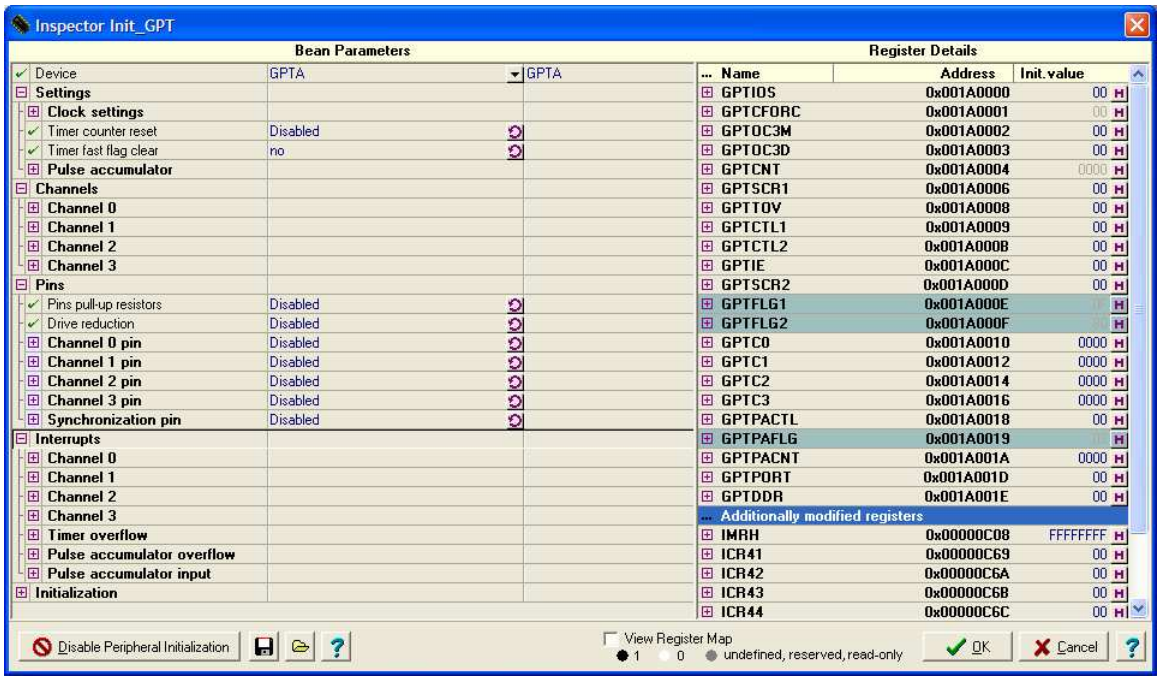

Figura 5.13: Configuració del GPT amb DI

5.3.3. RAD: Processor Expert

L'últim mode és el més avançat. El Processor Expert (PE) ens permet configurar els mòduls de la mateixa forma que el mode anterior i a més ens ofereix una llibreria per cada mòdul i un gestor de les interrupcions més senzill.

PE utilitza un paradigma de programació orientada a objectes per a organitzar el seu funcionament. A partir d'ara ja no parlarem de mòduls, sinó que parlarem de *beans*. Un *bean* és la configuració, les funcions de la seva llibreria associada i els *events* relacionats amb el mòdul.

Figura 5.14: Processor Expert

Quan vulguem activar un mòdul, el que farem és afegir el bean corresponent a aquest mòdul al projecte a través de l'assistent seleccionador de *beans* (figura 5.15). Això farà que la pròxima vegada que construïm el projecte, el PE afegeixi les funcions de la llibreria associada al mòdul actiu i també totes les instruccions de configuració.

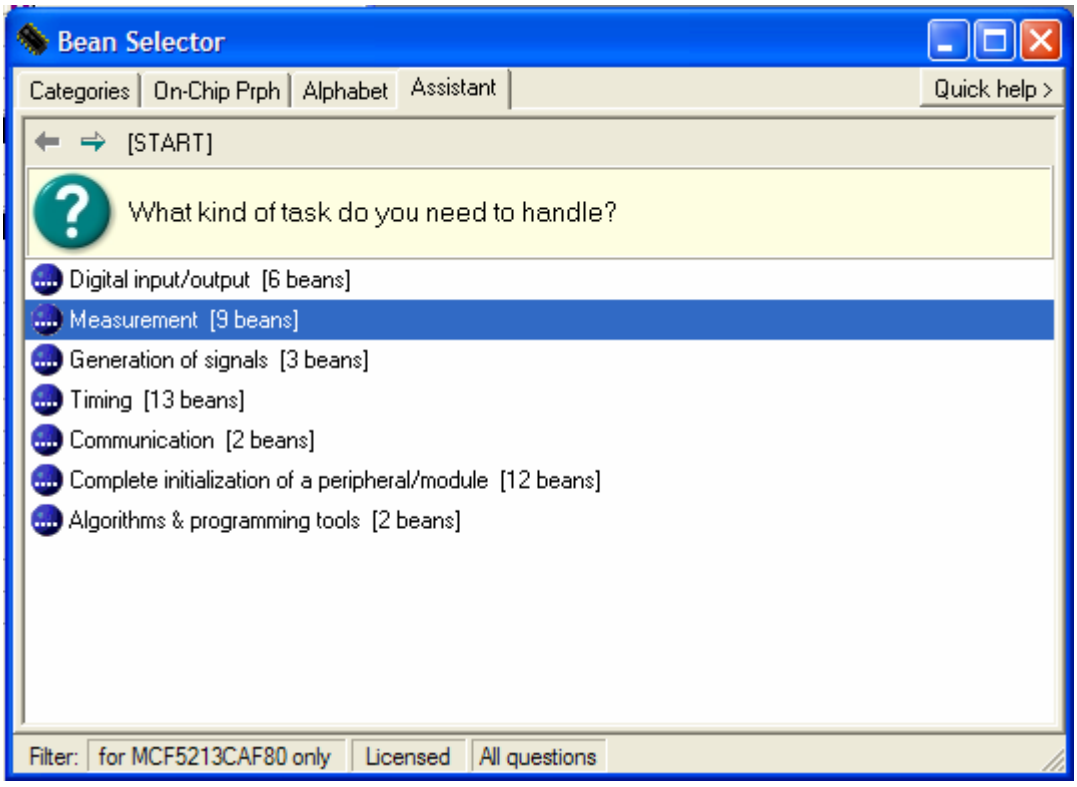

Figura 5.15: Bean Selector

El Processor Expert disposa del seu propi panell de funcionament en el gestor del projecte (figura 5.16). En aquest panell apareixeran els *beans* afegits al projecte amb les seves funcions disponibles i els possibles *events*.

Les funcions disponibles són aquelles funcions que han implementat els creadors del bean per tal d'interactuar directament amb les dades del nostre mòdul a més també d'oferir la possibilitat, en alguns casos, de modificar configuracions al llarg del transcurs de l'aplicació.

Podem trobar funcions tant diferents com captura de valors de registres, activació de sortides de senyal, enviament de dades, etc.

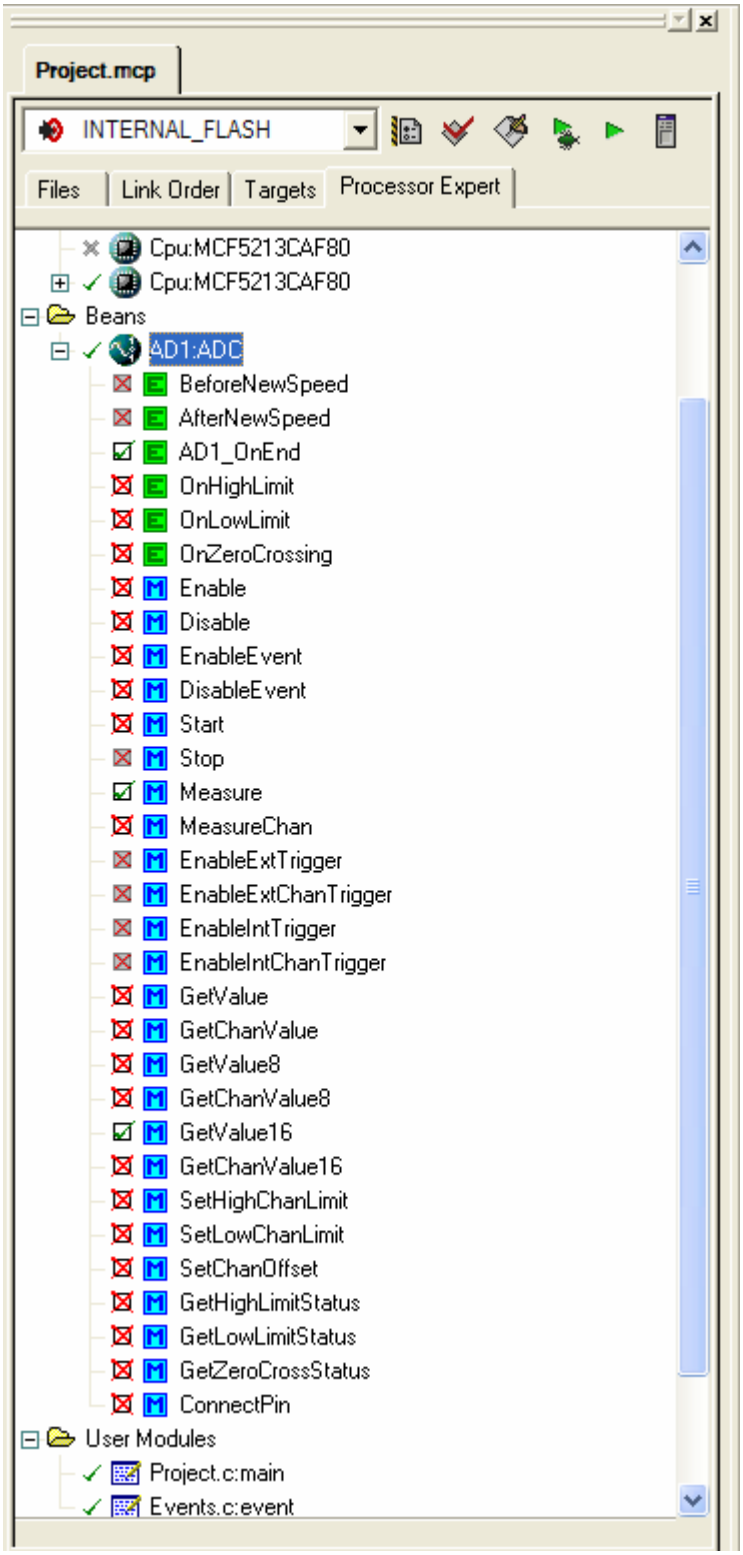

Figura 5.16: Funcions PE

Els events són les interrupcions que ens genera el mòdul, però expressats en forma de funció. Aquesta funció estarà buida, i

serà on podrem afegir el codi que vulguem per tal de tractar el succés. Trobem un exemple d'un event en el codi 2.

```
/*
** ===================================================================
**     Event       :  AD1_OnEnd (module Events)
**
**     From bean   :  AD1 [ADC]
**     Description :
**         This event is called after the measurement (which consists
**         of <1 or more conversions>) is/are finished.
**         The event is available only when the <Interrupt
**         service/event> property is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void AD1_OnEnd(void)
{
  /* Write your code here ... */
}
```

Codi 2: Funció d'un event

Un dels avantatges de treballar amb el PE és que podem activar o desactivar tant funcions com events. El fet d'activar o desactivar aquests ítems ens reduirà el codi generat per l'eina, i per tant ens permetrà encabir més codi d'usuari.

Un cop hem vist com es realitza l'accés a les dades i el control de les interrupcions, només ens queda veure com es realitza la configuració dels diferents mòduls.
El cas del Processor Expert serà molt similar al mòdul anterior. Es tracta d'un formulari amb només els camps que podem modificar i indicacions per a fer-ho correctament (figura 5.17). A més a més, comprovarà incompatibilitats entre els diversos paràmetres que configurem.

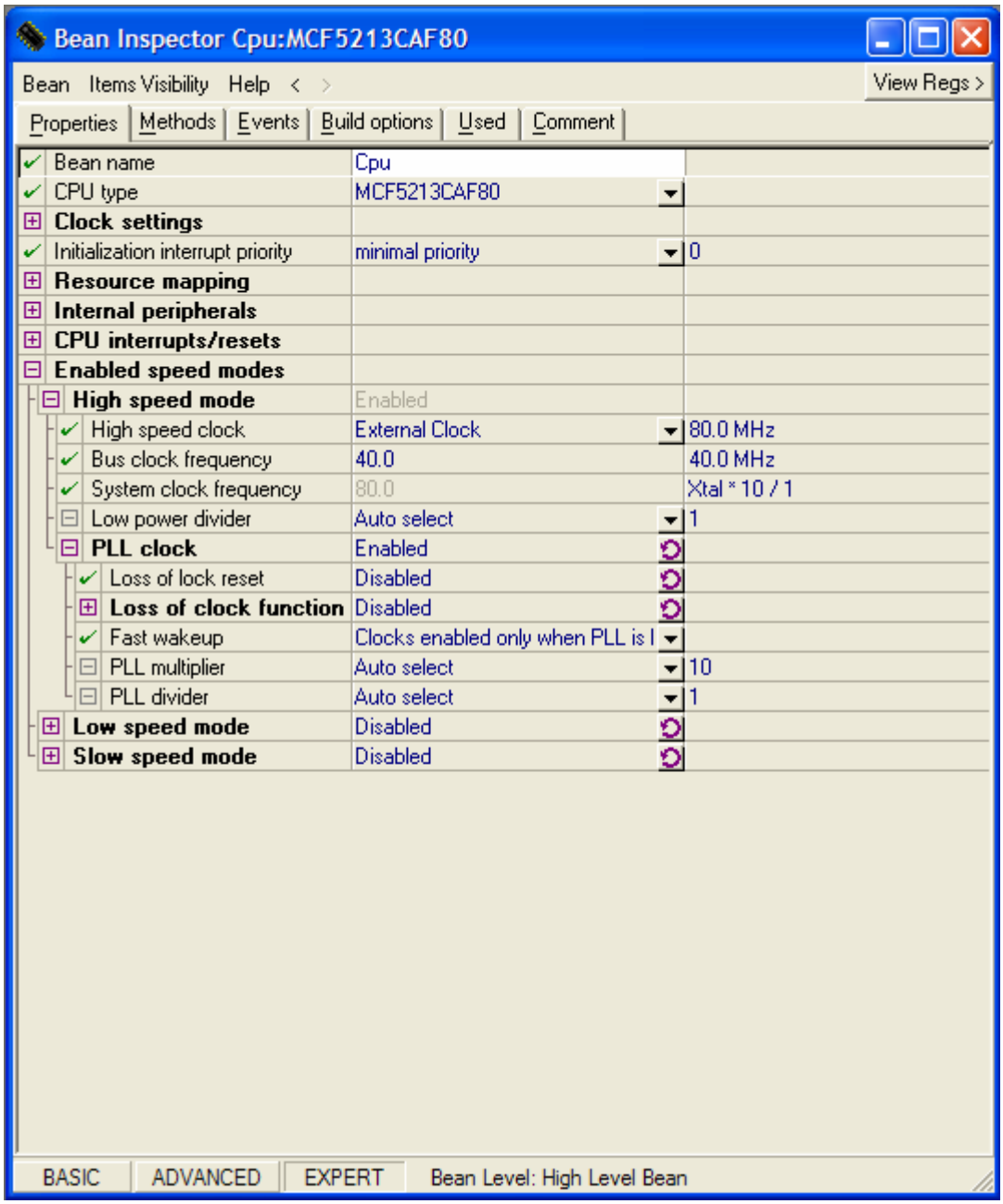

Figura 5.17: Configuració d'un bean

Finalment, només volem remarcar que el fet d'utilitzar el PE pot resultar molt útil en moltes situacions, però també hem de tenir en compte que moltes vegades, degut a la senzilla utilització que farem dels mòduls, és molt més eficient fer-ho directament a través de registres. Potser la configuració no és tant senzill, però la captura de dades i la interacció de forma dinàmica amb el mòdul

pot acabar sent molt senzilla si ho fem amb el Inicialitzador de Dispositius.

Per acabar, deixem aquí una taula comparativa (figura 5.18) entre el Processor Expert i el Device Initializator (Inicialitzador de Dispositius), per a poder observar en quins casos és beneficiós utilitzar un o altre.

Feature	Processor Expert	Device Initialization
Easy to use graphical IDE	✔	✔
Interactive design specifications of Freescale MCUs	✔	✔
Generated code	Peripheral Drivers	Initialization code only
Generated code languages	C	C
Peripheral Init Beans	✔	✔
Low Level Beans	✔	✘
High Level Beans	✔	✘
Project Configurations	✔	✘
User-friendly linker parameter file configuration	✔	✘
Generated code changes tracking	✔	✘
Timing settings in time units (seconds, bauds etc.)	✔	✘
Free beans library on the web	✔	✘
User beans creation	✔	✘

Figura 5.18: Comparativa RAD

6. Desenvolupament de l'Aplicació

El desenvolupament de l'aplicació és el procés d'anàlisi, disseny i codificació del programari que ens permetrà implementar tots els procediments que hem estudiat en l'apartat de tractament del senyal d'aquesta memòria.

Desenvolupar una aplicació funcional d'una forma eficient, sense errors i de forma sistemàtica és l'objectiu de l'Enginyeria del Software. Tot i tractar-se d'una aplicació embedida en un sistema electrònic, ens basarem en els principis d'aquesta enginyeria per tal de desenvolupar la nostra aplicació.

Bàsicament, voldrem que la nostra aplicació compleixi amb els requisits bàsics del bon software:

- De fàcil manteniment
- Econòmic
- Eficient
- Fiable
- Funcional
- Immediat
- Senzill

Per a satisfer aquests requisits ens basarem en un paradigma de desenvolupament de software que escollirem en el següent apartat.

6.1. Explicació de la Metodologia Emprada

En aquest apartat explicarem quina metodologia utilitzarem per a desenvolupar la nostra aplicació. El procés de software[1] són els passos i mètodes que ens permetran produir software de qualitat que compleixi els requisits exposats en l'apartat anterior.

L'enginyeria de software ens mostra diversos models de procés per a que apliquem el més convenient segons el projecte en el que estem treballant:

- Models Lineals
    - Model en Cascada
- Models Iteratius
    - Model Increments
    - Model DRA
- Models Evolutius
    - Model en espiral
    - Model concurrent
- Model basat en components
- Procés Unificat

No hi ha un procés que sigui millor que un altre, sinó que cadascun té els seus avantatges i els seus inconvenients. Haurem d'escollir, doncs, el que més s'adeqüi al nostre projecte. Per això haurem d'analitzar quines característiques té el nostre projecte.

Per una banda, aquest és un projecte on el software produït no canviarà gaire al llarg del temps. Hem de fer un sistema que llegeixi unes dades (que sempre estaran codificades de la mateixa

[1] Model de Desenvolupament de Software, segons quina bibliografia consultem.

forma), li apliqui unes correccions (que sempre seran les mateixes) i ho enviï per un bus adequat, aquest últim serà l'únic punt que podrà variar una mica. Una variació d'aquesta magnitud, però, segurament ens portarà al redisseny del circuit i l'entorn del microcontrolador i el sensor, per tant, no representarà un senzill canvi de requeriments, sinó un canvi de tot el sistema.

Per altra banda, les funcionalitats que ha d'implementar el nostre software són fàcilment modulables. De fet, la pròpia estructura del microprocessador ens divideix la feina en mòduls que s'han de configurar i executar. Podem, fàcilment, passar de l'estructura dels mòduls a les funcionalitats, el que ens encamina cap a un model iteratiu.

Així doncs, coneixent que el nostre sistema és molt estàtic i alhora fàcilment divisible en increments de mida adequada, sembla que la millor opció és adoptar el **model incremental**.

El model incremental combina els avantatges de la solidesa del model en cascada i de la capacitat de canvi dels models evolutius. El desenvolupament, fonamentalment, es basa en el model en cascada però iterant diverses vegades el cicle.

En cada iteració, anomenada increment, afegirem una funcionalitat més al sistema, passant d'un sistema molt simple al principi, amb només les funcionalitats principals, a un sistema complet, amb totes les funcionalitats que requereix el client.

El primer que cal fer és definir un esbós dels requeriments principals, és a dir, què farà el programa a grans trets. Un cop tenim clar què ha de fer el sistema, assignarem les diferents funcionalitats als increments, tenint en compte que són les funcionalitats principals les que s'han d'assignar als primers

increments, i les menys rellevants s'han de desenvolupar al final del procés.

Finalment dissenyarem l'arquitectura general del sistema, que quedarà invariable des d'aquest punt. És molt important no cometre errors en la definició de l'estructura, ja que aquests errors, en cas d'haver-se de corregir, podrien augmentar enormement el cost del desenvolupament.

Amb l'arquitectura ben definida, iniciarem les iteracions sobre el model en cascada, cada una de les quals donarà lloc a un increment amb una nova funcionalitat afegida.

Cada un dels increments es farà en 4 fases:

1. **Desenvolupar increment**: és on es dissenya i es codifica la nova funcionalitat. Normalment es realitza a part del projecte principal, per exemple, en una branca.
2. **Validar increment**: en aquesta fase haurem de comprovar el correcte funcionament de la nova funcionalitat, és a dir, de l'increment, sense la resta del projecte. Es poden utilitzar dades simulades i testos unitaris, si es considera necessari.
3. **Integrar increment**: ha arribat el moment d'integrar la nova funcionalitat al sistema, donant lloc a un nou increment que es pot lliurar al client. Com si d'un desenvolupament a través de reutilització de mòduls es tractés, caldrà ajuntar l'increment al projecte en curs.
4. **Validar sistema**: quan ja tenim el nou increment llest, haurem de comprovar que el sistema continua sent estable malgrat la funcionalitat afegida. Si es tracta de l'últim increment definit, es podrà ja realitzar una entrega final del producte, si no, s'inicia el desenvolupament del següent increment.

Al finalitzar cada increment, aquest s'entrega al client per a que el provi. Això significa que el client rep una primera versió

(anomenada *versió temprana*) del sistema molt aviat, i pot començar a provar-la i explorar els seus requeriments més fàcilment.

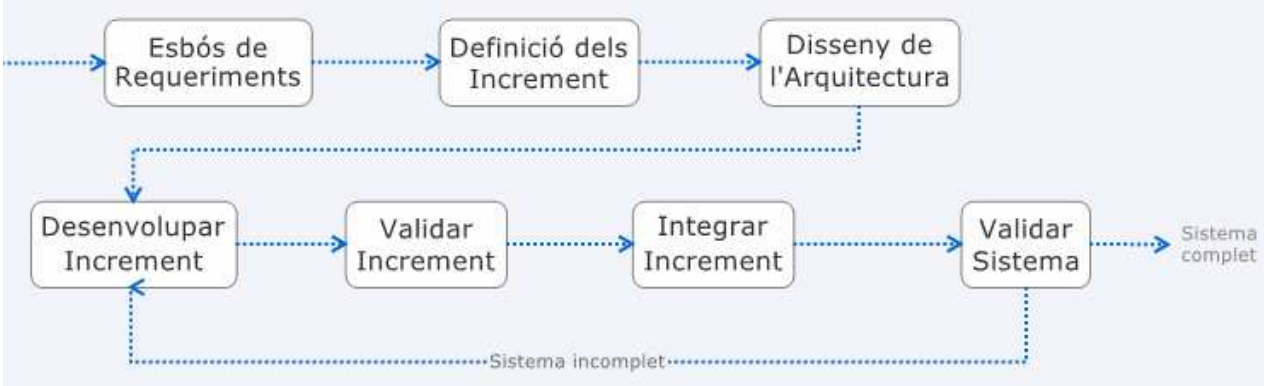

Figura 6.1: Model Incremental

Basar-se en aquest model de procés de software te diversos avantatges:

- Els clients no han d'esperar a l'últim increment del sistema. Poden començar a treure benefici de la pròpia *versió temprana*.
- Els clients poden utilitzar els primers increments per a adquirir consciència i coneixement del significat dels requeriments, i millorar la seva especificació.
- El risc de fallida del projecte és molt baix. Pot haver certes complicacions en l'entrega d'un increment, però el projecte acostumarà a acabar complet.
- Al implementar les funcionalitats més importants en els primers increments i integrar les més secundàries en aquests, és inevitable que les parts crítiques del sistema siguin les que més proves han de superar, donant lloc a una gran solidesa de l'estructura.

Presenta, també, alguns inconvenients que haurem d'intentar evitar en la mesura del possible:

- En certs projectes resulta difícil definir uns increments de mida adequada (no més de 20.000 línies de codi) i que implementin una funcionalitat completa.
- Aquelles funcionalitats utilitzades per diverses parts (o la totalitat) del sistema poden quedar mal definides fins que no s'han realitzat tots els increments, i en el sistema final, tenir una estructura poc ordenada.

En els següents apartats anirem descrivint el desenvolupament de la nostra aplicació a través d'aquest procés de software. Com indica la bibliografia[1] haurem de començar per una visió global de l'aplicació, amb un esbós de requeriments, l'assignació de funcionalitats als increments i l'arquitectura del sistema. Més endavant veurem cada iteració en detall.

[1] Ingeniería de Software (Sommervile, I).

6.2. Anàlisi de Requeriments

El primer a fer, per tal d'assegurar-nos que entenem l'objectiu del projecte, és elaborar una llista provisional de requeriments. Per a facilitar l'estructura dels requisits, hem decidit agrupar-los per temàtica:

- Tren de Polsos
    - El Sistema haurà de poder capturar l'amplitud de pols de 3 trens de polsos.
    - El Sistema haurà de donar l'amplitud del tren amb una precisió mínima de 10^{-6} segons.
- Temperatura
    - El Sistema haurà de capturar la temperatura de dos xips.
    - El Sistema haurà d'oferir una precisió de 0'2ºC.
- Bus de Dades
    - El Sistema haurà de transmetre les dades dels 3 eixos a través de I^2C.
    - El Sistema haurà de poder ser configurat a través de I^2C.
- Rendiment
    - El Sistema no pot afegir un retard superior a 250ms.
    - El Sistema ha d'oferir dades correctes d'acceleració al cap de com a molt 10 segons després del seu inici.
- Mida
    - La mida del software no pot superar els 256KB.
- Cost
    - Produir el software no pot costar més de 1000€.

6.3. Definició dels Increments

Abans d'assignar les funcionalitats als increments, haurem de definir aquestes funcionalitats del sistema. Cal recordar que primer haurem de desenvolupar les funcions crítiques, mentre deixem per al final aquelles que no afecten al als fonaments de l'aplicació.

Recordem breument els passos a realitzar per a poder enviar una mostra de l'acceleració:

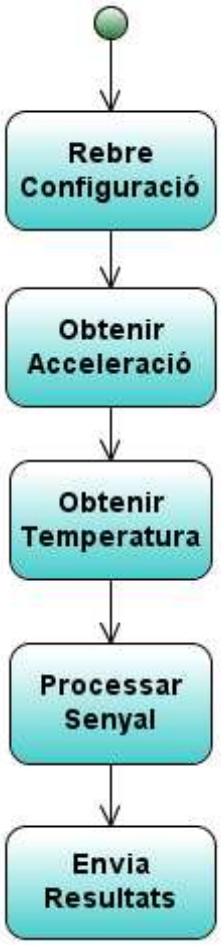

Figura 6.2: Passos del Sistema

Aquesta petita descripció del procediment ja ens indica quines poden ser les funcionalitats:

1. Configurar el sistema per I^2C.
2. Llegir d'un tren de polsos.
3. Llegir una senyal analògica.
4. Aplicar operacions matemàtiques.
5. Enviar dades per I^2C.

Si, a més a més, tenim en compte que ho hem de fer per a 3 eixos a través de dos sensors diferents, i que resulta molt més fàcil implementar-ho tot només per a un eix i després extrapolar-ho a la resta, ja podem definir els increments:

1. **Obtenir acceleració**: llegirem un tren de polsos i aplicarem les transformacions per a tenir el valor de l'acceleració.
2. **Obtenir temperatura**: capturarem el valor del senyal de temperatura i obtindrem el valor en graus d'aquesta.
3. **Realitzar la correcció i el filtrat del senyal**: aplicarem la correcció a l'acceleració basada en la temperatura. També aplicarem un filtre al senyal.
4. **Enviar dades per I^2C**: treballant amb dades fictícies de longitud 1 *byte*, enviarem missatges a un *master* del bus.
5. **Configurar i enviar dades reals per I^2C**: establirem una opció per tal de poder configurar el sistema amb només un missatge del *master* del bus. També començarem a treballar amb dades reals, havent de fraccionar-les, etc.
6. **Treballar amb 3 eixos i 2 sensors**: un cop tenim tot el sistema funcionant per a un eix, l'hem d'ampliar per a treballar amb els 3 eixos.

Abans de començar amb els increments, però, hem de definir l'arquitectura global del sistema, que no variarà al llarg del desenvolupament, si no és que s'ha comès un error molt greu.

6.4. Arquitectura del Sistema

Per a definir l'arquitectura del sistema ens basarem principalment en es mòduls amb els que treballarem al llarg de l'execució de l'aplicació. Aquests mòduls del microcontrolador són els que ja hem descrit en aquest document:

- Analog-to-Digital Converter (ADC)
- General Purpose Timer (GPT)
- Bus de Dades I^2C

A més a més, afegirem la unitat de processament central[1] que serà l'encarregada de realitzar els càlculs. Podem veure un diagrama de les seves relacions en la figura 6.3.

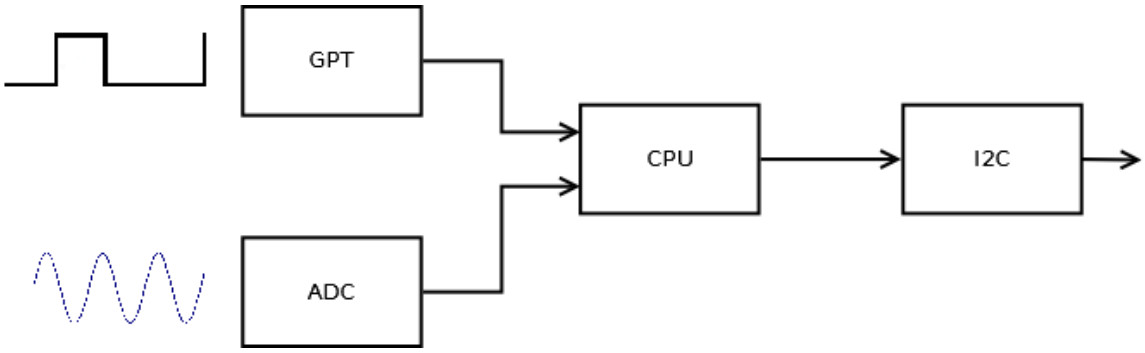

Figura 6.3: Diagrama de Mòduls

Sembla força intuïtiu emprar el GPT com a mòdul iniciador de la rutina, capturant el valor de l'ample de pols del tren de polsos que rebi del sensor cada 10ms.

Mentrestant, el mòdul ADC estarà contínuament transformant les dues senyals de temperatura d'analògica a digital. Cada transformació sobreescriu el resultat de l'anterior, per tant podem

[1] CPU, Central Processing Unit en anglès.

emprar aquest mòdul de forma continua sense perdre capacitat de memòria.

Hem decidit que sigui el *master* del bus qui demani les dades al nostre sistema, per tant anirem emmagatzemant en un registre la última dada corresponent a l'acceleració, i quan se'ns demani, la enviarem per I^2C.

Així doncs, aprofitarem les interrupcions que genera el GPT cada vegada que hi ha un canvi de flanc a la senyal per tal d'iniciar tot el procediment. En el cas de ser un flanc de pujada només guardarem les dades que pertoqui, mentre que si es tracta d'un flanc de baixada, obtindrem l'acceleració, la corregirem i la guardarem. Quan se'ns soliciti, enviarem l'últim valor de l'acceleració. Això ens permet ampliar una mica el diagrama anterior:

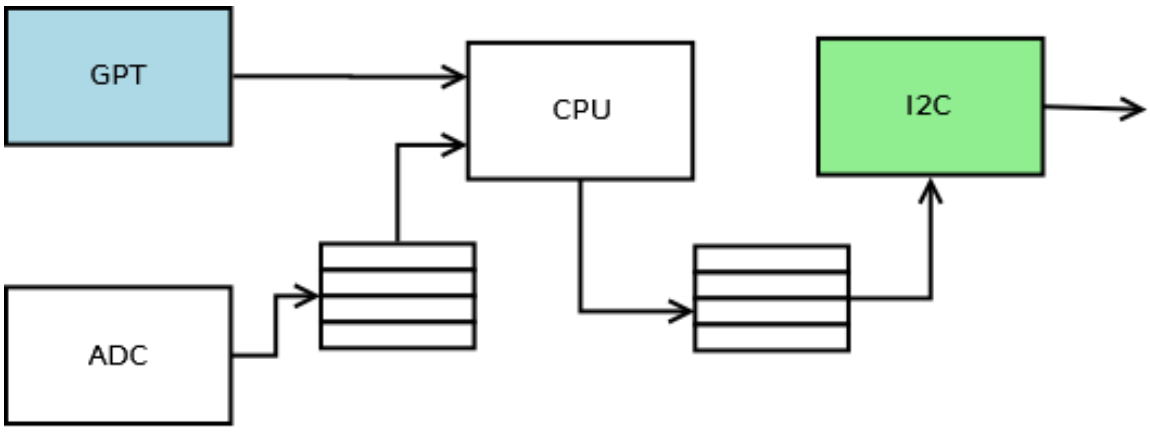

Figura 6.4: Diagrama de Mòduls Ampliat

On el mòdul marcat en blau (GPT) és l'iniciador de l'acció i el mòdul I2C és el que finalitza l'acció amb la transmissió de l'acceleració. Com podem veure, el sistema funciona contínuament,

encara que el *master* no hagi demanat dades. Hi hem afegit també uns símbols que representen els registres intermitjos que faciliten que el sistema treballi de forma continua.

6.5. Increments

Un cop hem definit els diferents increments en els que hem dividit l'aplicació, és el moment de detallar-los. Com ja hem explicat, en el model incremental, cada increment està format per 4 etapes:

1. Desenvolupament de l'Increment
   a) Anàlisi
   b) Codificació
2. Validació de l'Increment
3. Integració en el Sistema
4. Validació del Sistema

La fase de Desenvolupament de cada un d'aquests increments es divideix en 2 parts. La primera, l'anàlisi, servirà per a definir exactament què es farà en aquest increment. La segona serà la de codificació, on mostrarem el codi i les funcions més importants de la nostra aplicació.

Per a permetre una lectura més senzilla i contínua d'aquest document, hem decidit treballar en cada increment amb 3 fases:

- Anàlisi
- Codificació
- Validació

En general podem assumir que la validació de l'increment i del sistema es comproven en un mateix joc de proves, i que la integració en el sistema es defineix al llarg de la codificació.

La codificació consta de les configuracions dels mòduls i del codi. Aquest codi es distribuirà, segons l'estructura de projectes de CW, entre el programa principal (*main.c*) i el fitxer de successos (*events.c*).

6.5.1. Primer Increment

Anàlisi

En aquesta primera iteració el que hem de fer és llegir un tren de polsos, obtenir l'amplada del pols i transformar-la en el valor de l'acceleració.

Per a la captura del tren de polsos utilitzarem les interrupcions del bean GPT. El primer que haurem de fer és configurar el *bean* amb la major resolució i activar la interrupció per flanc de pujada. Com ja hem dit en el disseny de l'arquitectura del sistema, farem tot el tractament en aquesta interrupció.

Dins la interrupció hem de diferenciar dos situacions:

- **Flanc de pujada**: serà el cas en que la interrupció s'haurà activat per el flanc de pujada. En aquest cas ens trobem a l'inici del pols, per tant, l'únic que hem de fer és guardar el valor del temporitzador i posar l'*overflow* a zero.
- **Flanc de baixada**: en aquest cas és quan ha acabat el pols. Serà aquí on realitzarem tot el tractament. Haurem de calcular els cicles de rellotge[1] que han passat i transformar-ho a acceleració. Tenint en compte les dades de la configuració i l'equació d'acceleració que ens proporciona el fabricant, per a l'obtenció de l'acceleració haurem de calcular:

$$A(ticks) = ticks \cdot 0'00002 - 4 \qquad (6.1)$$

Per això haurem d'utilitzar el valor actual i anterior del temporitzador i el valor de l'*overflow*. La fórmula per a l'obtenció dels cicles de rellotge, serà:

$$ticks = (overflow + flanc\,baixada) - flanc\ pujada \qquad (6.2)$$

[1] Anomenarem *ticks* als cicles de rellotge, per treballar d'acord amb la bibliografia.

En ambdós casos, al iniciar el tractament del flanc corresponent hem de desactivar la interrupció pel mateix flanc i, al acabar el tractament, activar la interrupció pel flanc contrari.

Degut a la ràpida evolució del *overflow* i dels valors del temporitzador, hem decidit que el primer que farem en llençar la interrupció és guardar aquests valors, per a no perdre precisió, i que sigui tot el més fiable possible.

Sobre l'*overflow*, recordem que també tenim una interrupció que ens el permet gestionar. Cada vegada que es llanci aquesta interrupció, sumarem el valor de l'*overflow* (65535) a una variable global que ens el guarda. Com hem vist, posarem a zero aquesta variable a l'inici de cada pols.

Codificació

En l'apartat de codificació, el primer que hem de fer és configurar correctament el *bean*. Per això, l'afegirem al projecte i ens assegurarem que el temps del succés és el mínim (màxima resolució). També hem de definir el flanc de pujada com l'activador de la interrupció.

✓	Edge	rising edge	rising edge
⊟	**Interrupt service/event**	Enabled	
✓	Capture interrupt	INT_GPT_C0F	INT_GPT_C0F
✓	Capture priority	medium priority	level 4, priority within level 3
⊟	**Overflow support**	Enabled	
✓	Overflow interrupt	INT_GPT_TOF	INT_GPT_TOF
✓	Overflow priority	medium priority	level 4, priority within level 3
✓	Maximum time of event	1638.400 µs	high: 1638.400 µs

Figura 6.5: Configuració. 1r Increment

L'elaboració de codi, com a tal, es realitza únicament en el fitxer de tractament de les interrupcions. En aquest cas emprarem les dues interrupcions que hem citat.

Per a facilitar l'accés concurrent a les variables d'*overflow* i evitar crear les mateixes variables cada vegada que s'executi la interrupció, hem declarat aquestes variables com a globals.

```
uint32 ticks;
double accel;

unsigned short captura = 0;
unsigned short captura_temp = 0;
uint32 overflow = 0;
uint32 overflow_temp = 0;
```

Codi 3: Variables. 1r Increment

On *ticks* és la variable per emmagatzemar els cicles de rellotge transcorreguts en l'ample del pols, *accel* serà l'acceleració en g's, captura és el valor actual del temporitzador, *captura_temp* és el valor anterior del temporitzador, *overflow* és el valor actual del l'*overflow* i *overflow_temp* és el valor dinàmic de l'*overflow*.

Vegem primer el codi relacionat amb l'*overflow*. Com hem dit, l'únic que ha de fer és augmentar el valor de la variable acumuladora d'aquest *overflow* en la quantitat indicada (codi 4).

```
/*
** ===================================================================
**     Event       :  Cap1_OnOverflow (module Events)
**
**     From bean   :  Cap1 [Capture]
**     Description :
**         This event is called if counter overflows (only when the
**         bean is enabled - <Enable> and the events are enabled -
**         <EnableEvent>.This event is available only if a <interrupt
**         service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cap1_OnOverflow(void)
{
    overflow_temp += 65536;
}
```

Codi 4: Tractament de l'Overflow. 1r Increment

L'altra interrupció és la que ens indica un canvi en la senyal. Com podem veure en el codi 5, utilitzem la funció Cap1_GetCaptureValue per a obtenir el temps del GPT. Com hem dit, és el primer que farem per a obtenir una màxima precisió, igual que el valor de l'*overflow* en aquest mateix instant.

```
/*
** ===================================================================
**     Event       :  Cap1_OnCapture (module Events)
**
**     From bean   :  Cap1 [Capture]
**     Description :
**         This event is called on capturing of Timer/Counter actual
**         value (only when the bean is enabled - <Enable> and the
**         events are enabled - <EnableEvent>.This event is available
**         only if a <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cap1_OnCapture(void)
{
    /* recollim les dades: temps i overflow */
    Cap1_GetCaptureValue(&captura);
    overflow = overflow_temp;

    if (testReg8Bit(GPTCTL2, EDG0A)) {      /* flanc de pujada */

        /* desactivem la interrupcio per flanc de pujada */
        clrReg8Bit(GPTCTL2, EDG0A);

        overflow_temp = 0;
        captura_temp = captura;

        /* activem la interrupcio per flanc de baixada */
        setReg8Bit(GPTCTL2, EDG0B);

    } else {                                /* flanc de baixada */

        /* desactivem la interrupcio per flanc de baixada */
        clrReg8Bit(GPTCTL2, EDG0B);

        ticks = (overflow + captura) - captura_temp;
        accel = (ticks*0.00002) - 4;

        /* activem la interrupcio per flanc de pujada */
        setReg8Bit(GPTCTL2, EDG0A);
    }
}
```

Codi 5: Captura. 1r Increment

Validació

Per tal de validar aquest increment, el que farem és definir un vector on guardarem 1000 mostres de l'acceleració. Iniciarem l'aplicació amb el depurador, per tal de poder comprovar el valor d'aquest vector un cop estigui emplenat.

Un cop arrancada l'aplicació i el microcontrolador estigui prenent mostres, alternarem la posició de l'acceleròmetre, d'horitzontal a vertical. Els resultats guardats en el vector mostraran aquest comportament.

Val a dir que no avaluem la magnitud real d'aquests valors, només el canvi. No te sentit avaluar si la senyal és bona o dolenta, ja que encara no s'ha tractat.

6.5.2. Segon Increment

Anàlisi

En aquest segon increment ens proposem afegir suport per a la lectura de la temperatura a través del mòdul Analògic-Digital. Per a no interferir en la pila d'interrupcions que haurà de gestionar el microcontrolador, hem decidit no utilitzar les interrupcions d'aquest mòdul.

Treballarem amb el ADC a través d'una configuració inicial i després accedint directament al registre de resultats quan vulguem recuperar les dades.

Aquesta configuració haurà de ser la següent:

- Freqüència ADC: 5MHz
- Mode Captura: Bucle Paral·lel Simultani
- Voltatges de referència: interns
- Iniciar Captures ADC: sí
- Apagar ADC: no

El procediment de captura serà senzill. Quan estiguem en el flanc de baixada de la interrupció del GPT, llegirem dels registres de resultats corresponents. Sobre el valor d'aquests registres haurem de realitzar dues operacions:

1. Desplaçar a la dreta el valor 4 posicions, per a compensar el fet que en el registre el resultat es troba justificat a l'esquerra.
2. Transformar el valor de l'escala a volts. En aquest cas, al emprar una diferència de voltatges de referència de 5v, el valor de cada esglaó és de 5/4096. Per tant, per obtenir el valor del voltatge real hem de multiplicar l'esglaó per aquesta magnitud:

$$V(e) = e \cdot \frac{5}{4096} \tag{6.3}$$

3. Transformar els volts a temperatura. Per a fer-ho ens basarem el les especificacions del sensor, que parlen d'aquest valor. Un cop calibrat el termòmetre[1], amb el valor de b correcte, podrem aplicar la següent funció:

$$t(v) = v \cdot 200 + b \tag{6.4}$$

Codificació

En aquest cas hem d'afegir un *bean* d'inicialització del sistema ADC: el *bean* Init_ADC. En aquest *bean* aplicarem la configuració que hem definit en l'apartat anterior:

Clock setting	
Clock divisor select value	divide by 8
ADC frequency	5.0000 MHz
Stop Mode 0	No
Stop Mode 1	No
ADC Mode	Loop parallel
Parallel mode	Simultaneous
Trigger mode 0	SYNC input or START bit
Trigger mode 1	SYNC input or START bit
High volt. ref. source	Internal
Low volt. ref. source	Internal
A/D channels	
Pins	
Interrupts	
Initialization	
Start ADC Conversion 0	Yes
Start ADC Conversion 1	Yes
Power Down ADC 0	No
Power Down ADC 1	No
Power Down Voltage Referenc	No
Auto Power-Down	Disabled
Power-up Delay	13
Auto Standby	Disabled
Call Init method	yes

Figura 6.6: Configuració. 2n Increment

[1] Trobarem una explicació de com fer-ho en l'annex B d'aquest document.

Sobre el codi a modificar, només caldrà afegir unes sentències que ens capturin el valor del registre en qüestió i ens calculin la temperatura corresponent després del càlcul de l'acceleració.

Primer haurem de declarar les variables globals adequades:

```
int regtemp = 0;
double vtemp = 0;
double t = 0;
double b = -223.74;
```

Codi 6: Variables. 2n Increment

I després, inserir les línies del càlcul de la temperatura després de l'obtenció de l'acceleració:

```
/* obtenim la temperatura */
regtemp = getReg16(ADRSLT0)>>4;
vtemp = regtemp*0.001220703125;
t = vtemp*200 + b;
```

Codi 7: Captura. 2n Increment

Validació

Per a comprovar el correcte funcionament de la captura de la temperatura, el que farem és capturar la mateixa senyal amb el microcontrolador i amb una targeta de captura de dades de la que disposem al laboratori. Comparant els resultats es pot apreciar que la targeta de captura ofereix més definició, però globalment el resultat és acceptable.

També aprofitem per a fer el mateix que en l'increment anterior per a verificar que aquesta nova funcionalitat no afecta a la captura de l'acceleració. Tot funciona correctament.

6.5.3. Tercer Increment

Anàlisi

En aquest increment haurem d'afegir les correccions a l'acceleració i el filtre de les dades. Els passos a realitzar són els següents:

1. Corregir el factor de sensibilitat, per el que haurem de multiplicar el valor de l'acceleració per la relació entre la temperatura actual i la temperatura ideal:

$$A_{compensada}(t) = A \cdot \frac{t^{2'81}}{8'9776 \cdot 10^{6}} \qquad (6.5)$$

2. Corregir la desviació. Per això haurem d'avaluar un polinomi de grau dos i restar-li a l'acceleració:

$$A_{compensada}(t) = A - (a + b \cdot t + c\, t^{2}) \qquad (6.6)$$

3. Filtrar la senyal resultant. Haurem de definir una rutina que ens elabori els càlculs adequats per aplicar el filtre.

Hem d'anar amb compte amb l'estructura d'aquests procediments. Si en els increments anteriors el tractament eren només un parell d'instruccions, ara el procediment de filtrat és més complex, per tant, en compromís amb el paradigma de programació descendent, el implementarem en una funció.

Codificació

En aquest cas no haurem de configurar cap mòdul, ja que ara només emprarem la CPU per a realitzar càlculs. Com en els altres increments, declararem les variables globals necessàries.

```
double h[50] = {...};
double R[50] = {0};
double p1 = 2.121e-005;
double p2 = 0.002689;
double p3 = -0.01996;
```

Codi 8: Variables. 3r Increment

On *h* és el vector on guardarem els coeficients del filtre[1], *R* és el vector que ens emmagatzemarà els últims 50 valors, i *p1*, *p2* i *p3* són els coeficients de la corba de correcció de la desviació.

El primer, és afegir el càlcul de les compensacions a la interrupció, just després de l'obtenció de les dades i després llançarem la funció que aplica el filtre:

```
/* corregim les dades */
accel = accel*((t^2.81)/8977600)-(p1*t*t+p2*t+p3);
accel = filtrar(accel);
```

Codi 9: Captura. 3r Increment

Finalment, la funció que aplica el filtre és:

```
double filtrar(double mostra) {
    double s=0;
    int i;

    for(i=49; i>0; i--){
        R[i]=R[i-1];
        s=s+R[i]*h[i];
    }

    R[0]=mostra;
    return s+R[0]*h[0];
}
```

Codi 10: Funció de Filtrat

Validació

Per a comprovar el correcte funcionament del sistema un cop hem afegit aquest mòdul, el que farem és enviar les dades un cop obtingudes i tractades a través del port UART a l'ordinador. D'aquesta manera podem rebre dades en temps real sobre el que està computant el microcontrolador.

[1] Podem trobar els coeficients del filtre a l'annex E d'aquest document.

Podríem haver-ho fet també amb el depurador, però per a saber si el temps de resposta del sistema continuava estant en el rang admissible, després d'afegir les operacions, necessitàvem conèixer les dades en temps real.

6.5.4. Quart Increment

Anàlisi

Un cop ja hem obtingut la dada correcta d'acceleració, el que hem de fer és enviar-la. Com ja hem comentat, emprarem el mòdul de configuració I^2C, però haurem d'interactuar amb els registres per tal d'enviar les dades.

La crida d'aquesta funcionalitat, però, es farà a través de la interrupció que gestiona I2C, per tant, el codi per a enviar les dades ha d'estar allí. L'acceleració, però, l'obtindrem de la variable global d'acceleració.

Primer el que haurem de fer és configurar el mòdul corresponent. En aquest cas haurem d'especificar:

- Freqüència
- Si som Mestre o Esclau
- Si hem de rebre o enviar dades
- Bit ACK
- ID
- Nom de la interrupció que gestionarà I^2C

En aquest increment, no treballarem amb dades reals, només enviarem un paquet de 8 bits cada vegada que rebem un paquet amb el nostre ID:

Codificació

En aquest cas, la configuració del mòdul quedarà de la següent manera:

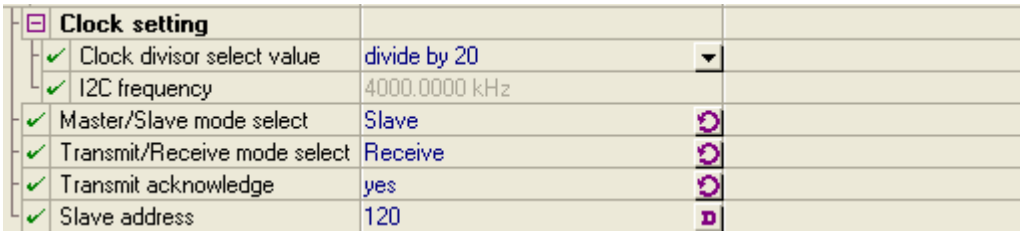

Figura 6.7: Configuració 4t Increment

El que haurem de fer, també és implementar la interrupció que es cridarà cada vegada que rebem un paquet amb el nostre ID:

```
/*
** ===================================================================
**     Interrupt handler : I2CRoutine
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__declspec(interrupt) void I2CRoutine(void)
{

    if(testReg8Bit(I2SR, IAAS) == 1){   /*hem estat adreçats*/

        if(testReg8Bit(I2SR, SRW) == 1){
            setReg8(I2DR, 0x03);
            while(testReg8Bit(I2SR, ICF) == 0);
        }

    }

    /* tornem a activar la interupcio */
    clrReg8Bit(I2SR, IFF);
}
```

Codi 11: Interrupció I2C. 4t Increment

En aquest codi el que fem és, primer, comprovar si hem estat adreçats com esclaus. Després, mirem si el *master* ens demana que escrivim al bus. De ser així, escrivim una dada i esperem a que aquesta hagi sigut enviada. Abans d'abandonar la interrupció, la tornem a activar.

Validació

Per a validar el correcte funcionament de I^2C el que hem fet és configurar una altra targeta d'avaluació per a que es comuniquin entre elles. S'ha comprovat que hi ha trànsit en el canal, i que les dades que s'envien són correctes.

6.5.5. Cinquè Increment

Anàlisi

En aquest increment, volem incorporar una nova funcionalitat al bus I^2C que ens permetrà configurar el nostre sistema mitjançant un codi i, a més, volem començar a treballar amb dades reals.

Al treballar amb dades reals, ens trobem amb el problema que són del tipus *double*, i no es poden enviar de forma nativa a través d'un canal que només admet l'enviament de *byte* a *byte*.

Haurem de descompondre la variable en 4 *bytes* per a poder enviar-los a través d'I^2C i reconstruir-la en el receptor. L'estudi de com fer una descomposició d'aquestes el trobem en l'annex D. Un cop tinguem les dades descomposades, les copiarem al registre corresponent de I^2C i les enviarem.

El funcionament bàsic per a la configuració serà el següent: al rebre un paquet amb un *byte* de dades, el sistema el llegirà i segons el valor del *byte*, aplicarà una configuració o una altra[1].
El protocol ja estableix una taula amb diferents possibles configuracions, per tant el que hem de fer és llegir aquest valor i modificar els registres segons la configuració.

Donat que en aquesta fase del desenvolupament encara no estan definides totes les possibles configuracions del sistema, hem deixat la plataforma preparada, però la funció que les ha d'aplicar no està encara implementada. S'haurà de fer quan s'hagin definit les configuracions.
Així, el que farem en aquesta fase és afegir una opció a la interrupció de I2C que, en cas de rebre un paquet de dades, cridi

[1] Podem trobar el protocol definit en l'annex C d'aquest document.

una funció i li passi per paràmetre aquestes dades. Serà la funció la que implementi la configuració.

```
__declspec(interrupt) void I2CRoutine(void)
{

    if(testReg8Bit(I2SR, IAAS) == 1){   /*hem estat adreçats*/

        if(testReg8Bit(I2SR, SRW) == 1){

            /* descomposem accel */
            byte *num1, *num2, *num3, *num4;
            num1 = ((byte*)(&accel));
            num2 = ((byte*)(&accel)+1);
            num3 = ((byte*)(&accel)+2);
            num4 = ((byte*)(&accel)+3);

            /* enviem el primer */
            setReg8(I2DR, num1);
            while(testReg8Bit(I2SR, ICF) == 0);

            /* enviem el segon */
            setReg8(I2DR, num2);
            while(testReg8Bit(I2SR, ICF) == 0);

            /* enviem el tercer */
            setReg8(I2DR, num3);
            while(testReg8Bit(I2SR, ICF) == 0);

            /* enviem el quart */
            setReg8(I2DR, num4);
            while(testReg8Bit(I2SR, ICF) == 0);

        } else {
            aplicarConfiguracio(getReg16(I2DR));
        }

    }

    /* tornem a activar la interupcio */
    clrReg8Bit(I2SR, IFF);
}
```

Codi 12: Interrupció I2C. 5é Increment

Codificació

El que hem de fer, doncs, és modificar la interrupció i afegir un condicional que detecti si hem rebut dades:

- Si hem estat adreçats per a enviar dades, descompondrem la variable *accel* en 4 bytes i els enviarem. Després d'enviar cada *byte*, esperarem a que el bus torni a estar lliure.
- Si hem estat adreçats per a rebre dades, en canvi, llegirem el registre i passarem el seu valor a la funció aplicarConfiguracio().

Podem veure el codi d'aquesta interrupció en el codi 12.

Validació

La validació completa d'aquest increment no es pot realitzar fins que no s'hagi construït la taula de configuracions. De totes formes, el que hem fet és llençar el programa en mode de depuració i hem comprovat, si al rebre un *byte* de dades entra a la funció que haurà de configurar el sistema.

6.5.6. Sisè Increment

Anàlisi

Ens trobem ja en el sisè i últim increment. Aquest increment el que ha de fer és aplicar el que ara hem aconseguit fer per a un sol eix, als tres.

Aquesta ampliació implica treballar amb 3 trens de polsos i amb dos senyals de temperatura. Un dels punts delicats pot ser que el propi microcontrolador no treballi a suficient freqüència per a poder capturar totes les senyals. Fem uns petits càlculs.
Si tenim 3 trens de polsos a 100Hz, vol dir que cada segon tenim $3 \cdot 100 \cdot 2 = 600$ interrupcions. Tenint en compte que el nostre processador treballa a 80MHz, això és, 80000 operacions per segon, tenim $80000/600 \approx 130$ instruccions per a tractar cada interrupció. Tenint en compte el filtre d'ordre 50 i les altres transformacions, no disposem de gaire marge.

El procediment a seguir per a aplicar el mètode als 3 eixos implica:

- Afegir 1 canal al ADC
- Afegir 2 *beans* de GPT, amb les seves funcions
- Afegir 2 vectors de coeficients
- Afegir 2 funcions de filtrat més
- Afegir el tractament de la segona senyal del primer xip
- Afegir els coeficients d'ajust de la corba del segon xip
- Afegir el tractament de correcció de l'altre xip
- Afegir 2 enviaments més al I^2C

Les configuracions, però, seran iguals a les que ja hem vist.

Codificació

Afegirem doncs, el canal al ADC i els nous *beans* de GPT. El CW ja ens els anomenarà de forma diferent, així com les funcions per accedir als seus valors i events:

Figura 6.8: Beans 6è Increment

En quant a les variables, les haurem de posar per triplicat en el cas de l'acceleració, i per duplicat en les relatives a les característiques intrínseques de cada sensor d'acceleració. Això ens porta a una gran quantitat de variables:

```
uint32 ticks1;
uint32 ticks2;
uint32 ticks3;
double accel1;
double accel2;
double accel3;

unsigned short captura1 = 0;
unsigned short captura2 = 0;
unsigned short captura3 = 0;
unsigned short captura_temp1 = 0;
unsigned short captura_temp2 = 0;
unsigned short captura_temp3 = 0;
uint32 overflow1 = 0;
uint32 overflow2 = 0;
uint32 overflow3 = 0;
uint32 overflow_temp1 = 0;
uint32 overflow_temp2 = 0;
uint32 overflow_temp3 = 0;
```

```
int regtempA = 0;
int regtempB = 0;
double vtempA = 0;
double vtempB = 0;
double tA = 0;
double tB = 0;
double bA = -223.74;
double bB = -223.74;

double h[50] = {};
double R1[50] = {0};
double R2[50] = {0};
double R3[50] = {0};
double p1A = 2.121e-005;
double p2A = 0.002689;
double p3A = -0.01996;
double p1B = 2.121e-005;
double p2B = 0.002689;
double p3B = -0.01996;
```

Codi 13: Variables. 6è Increment

Per a l'*overflow*, enlloc de només tenir una variable que el guardi, n'haurem de tractar amb 3:

```
/*
** ===================================================================
**     Event       :  Cap1_OnOverflow (module Events)
**
**     From bean   :  Cap1 [Capture]
**     Description :
**         This event is called if counter overflows (only when the
**         bean is enabled - <Enable> and the events are enabled -
**         <EnableEvent>.This event is available only if a <interrupt
**         service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cap1_OnOverflow(void)
{
    overflow_temp1 += 65536;
    overflow_temp2 += 65536;
    overflow_temp3 += 65536;
}
```

Codi 14: Tractament Overflow. 6è Increment

També haurem de fer 3 funcions per a filtrar la senyal, una per cada eix:

```
double filtrar1(double mostra) {
    double s=0;
    int i;

    for(i=49; i>0; i--){
        R1[i]=R1[i-1];
        s=s+R1[i]*h[i];
    }

    R1[0]=mostra;
    return s+R1[0]*h[0];
}
```

Codi 15: Funció Filtre. 6è Increment

La funció que gestioni la interrupció del GPT, haurà de treballar només amb l'*overflow*, les correccions i el filtre que li correspongui:

```
void Cap1_OnCapture(void)
{
    /* recollim les dades: temps i overflow */
    Cap1_GetCaptureValue(&captura1);
    overflow1 = overflow_temp1;

    if (testReg8Bit(GPTCTL2, EDG0A)) {      /* flanc de pujada */

        /* desactivem la interrupcio per flanc de pujada */
        clrReg8Bit(GPTCTL2, EDG0A);

        overflow_temp1 = 0;
        captura_temp1 = captura1;

        /* activem la interrupcio per flanc de baixada */
        setReg8Bit(GPTCTL2, EDG0B);

    } else {                                /* flanc de baixada */

        /* desactivem la interrupcio per flanc de baixada */
        clrReg8Bit(GPTCTL2, EDG0B);

        ticks1 = (overflow1 + captura1) - captura_temp1;
        accel1 = (ticks1*0.00002) - 4;

        /* obtenim la temperatura */
        regtempA = getReg16(ADRSLT0)>>4;
        vtempA = regtempA*0.001220703125;
        tA = vtempA*200 + bA;

        /* corregim les dades */
        accel1 = accel1*((tA*2.81)/8977600)-(p1A*tA*tA+p2A*tA+p3A);
        accel1 = filtrar1(accel1);

        /* activem la interrupcio per flanc de pujada */
        setReg8Bit(GPTCTL2, EDG0A);
    }
}
```

Codi 16: Captura. 6è Increment

Sobre l'enviament de dades a través d'I2C, només haurem de modificar la funció per a que transmeti els 3 valors, un radere de l'altre.

Validació

Aquest últim increment encara no ha estat validat a l'hora d'escriure aquesta memòria. Queda pendent, doncs, crear un petit sistema en una targeta que rebi les dades dels 3 eixos, les reconstrueixi i les mostri, de tal forma que puguem comprovar si s'ha rebut bé o no.

7.Valoració de Costos

En qualsevol projecte hi ha d'haver una valoració de costos per a que es prengui consciència del preu que s'ha de pagar per a desenvolupar o produir un sistema.

Podem trobar, però, dos grans tipus de projectes. En el primer, l'orientat a l'empresa i al mercat laboral, ha de contenir una valoració precisa, un pressupost que és vinculant i s'ha de complir.

En el segon tipus de projectes, els orientats a la investigació, aquesta avaluació no és tant un cost sinó una inversió d'hores i esforços d'un equip de treball.

Per aquest motiu, hem decidit no incloure una valoració de costos econòmica en aquest document, però sí una valoració temporal i un llistat d'equipament de laboratori emprat.

7.1. Equipament Utilitzat

Per al desenvolupament d'aquest projecte s'ha tractat amb equipaments de divers tipus. Des dels exclusivament electrònics, als informàtics, passant per materials de construcció. Obviant el material fungible utilitzat, l'equipament bàsic és:

- Targeta d'avaluació MCF5213EVB
- 2 Sensors MXD2125G
- Sonda de temperatura
- Generador de Funcions
- Oscil·loscopi
- Font d'alimentació
- Protoboard
- Soldador d'aire
- Aire comprimit
- Poliextiré expandit
- Ordinador amb Windows XP i:
    - CodeWarrior IDE
    - ColdFire Init
    - TeraTerm
    - MatLab

Tot aquest material no ha tingut un cost directe sobre el desenvolupament del projecte perquè, com ja hem dit, es tracta d'un projecte d'investigació, i disposàvem d'aquest material ja al laboratori.

7.2. Temporització

A banda de l'equipament i del material necessari, acostuma a ser més important encara una estimació del temps que s'ha invertit en el desenvolupament del projecte.

No estem parlant només de les hores que s'han dedicat a programar l'aplicació, sinó totes les hores que s'han dedicat a la recerca d'informació, a la lectura de manuals, a l'elaboració de proves i prototips, etc. Representen en conjunt moltes més hores que les dedicades purament a la implementació, ja que aquesta només es realitza quan ja s'han adquirit els coneixements necessaris.

Hem elaborat una taula amb les hores que havíem estimat que invertiríem i les que realment hem invertit. Ho hem indicat en setmanes per a facilitar la comprensió temporal. Considerem que una setmana són, aproximadament, 25 hores de dedicació.

Tasca	**Temps Previst**	**Temps Invertit**
Tria components	1 setmana	1 i ½ setmanes
Tractament del Senyal	5 setmanes	8 setmanes
Documentació ColdFire	3 setmanes	3 setmanes
Experimentació CodeWarrior	5 setmanes	8 setmanes
Disseny de l'Aplicació	2 setmanes	2 setmanes
Codificació de l'Aplicació	2 setmanes	5 setmana
Validació de l'Aplicació	1 setmana	1 setmana
TOTAL	19 setmanes	28 i ½ setmanes

Figura 7.1: Taula Temporització

Cal fer uns petits aclariments sobre les disparitats de setmanes entre el previst i l'invertit. Aquest és el primer projecte que realitzem de programació embedida en microcontroladors. Per això hem tingut alguns problemes.

El que pensàvem que havia de ser un entorn molt senzill de dominar, el CodeWarrior, ha acabat sent força més difícil. A més, hem realitzat petits projectes breument documentats per tal de deixar un fons de coneixement al laboratori. Això ha fet que la estimació, a priori generosa, de 5 setmanes hagi estat sobrepassada a la pràctica.

Una situació similar hem vist amb el tractament del senyal. Mai ens havíem enfrontat amb un tractament del senyal d'un sensor del que no coneixíem res. Per això, tot hi haver trobat suport del propi fabricant, el temps dedicat al tractament del senyal, entre formació i disseny, ha estat força més gran que el s'havia estimat.

Aquests errors, com diem, són deguts a la poca experiència en realitzar investigacions d'aquest tipus.

8. Conclusions i treball futur

Ara, després d'haver dut a terme tots i cada un dels passos detallats en aquest document, ha arribat el moment de donar un cop d'ull als objectius que ens plantejàvem a l'inici d'aquest projecte i comprovar quina ha esta la feina feta.

Vam iniciar aquesta aventura amb la idea de dissenyar un sistema que mesurés en temps real l'acceleració a la que està sotmès un dispositiu mòbil autònom en qualsevol dels 3 eixos ortogonals. En resum, volíem estudiar un sistema que ens permetés tenir un acceleròmetre tridimensional, però força precís.

A la finalització d'aquest projecte disposem del disseny i implementació d'un sistema que captura les dades d'acceleració i els aplica un tractament per tal de millorar la seva precisió i la seva estabilitat. Aquest sistema, com qualsevol sistema, afegeix un retard acceptable, que no ens fa perdre massa correcció en el temps.

En quant als objectius secundaris, hem anat assolint-los un a un al llarg del desenvolupament d'aquest projecte. Començant per els relacionats amb la tria, i acabant amb els de l'ús del CodeWarrior. Considerem que aquests petits objectius ens han guiat al llarg del procés, i els valorem molt positivament.

Podem dir, doncs, que els objectius d'aquest projecte han estat assolits. Però no està tot fet. Encara hi ha força coses que es poden millorar per a obtenir un sistema encara més estable:

- Existeix un mètode que permet corregir la resposta en freqüència d'un acceleròmetre. No aporta grans millores, segons sembla, però és un camp que s'ha d'estudiar.
- El protocol I^2C que hem implementat en la nostra aplicació, no fa ús de la possibilitat de notificar a l'emissor que hem rebut el seu missatge. Això pot ajudar en gran mesura a estabilitzar i comprovar el correcte funcionament de tot el sistema.
- També s'ha de treballar en l'especificació de la taula de possibles configuracions que pot adoptar el nostre sensor, per a poder oferir-les al gestor del dispositiu.
- Un altre punt que es pot millorar, tot i que no afecta al rendiment, és l'ús de llibreries optimitzades per a la nostra aplicació, és a dir, no carregar totes aquelles funcions que sabem que no necessitarem.

També creiem oportú afegir que al realitzar un projecte d'investigació sobre un camp que no havíem tractat abans, hem après moltes coses que, d'altra forma, desconeixeríem.

9. Bibliografia

- Proakis J., Manolakis D. *Tratamiento digital de señales*. 3a Edició. Ed. Prentice Hall.
- Claria F. *Sistemas de Control Digital*. Quaderns EUP nº9.
- Pressman R. *Ingeniería del Software. Un enfoque práctico*. 6a Edició. Ed. McGrawHill.
- Sommerville I. *Ingeniería del Software*. 7a Edició. Ed. Pearson.
- Freescale Semiconductor Inc. *MCF5213 ColdFire Integrated*
- *Microcontroller Reference Manual.*
- Freescale Semiconductor Inc. *MCF521X Family Fact Sheet*.
- Freescale Semiconductor Inc. *ColdFire Family Programmer's Reference Manual*.
- Freescale Semiconductor Inc. *CodeWarrior Development Studio IDE 5.7 User's Guide*.
- Parallax Inc. *Memsic 2125 Dual-Axis Accelerometer Datasheet*.
- MEMSIC Inc. *MXD2125G/M/N/H Datasheet Rev.E.*
- MEMSIC Inc. *Thermal Accelerometer Temperature Compensation* AN-00MX-02.

Annex A. Codi Font

Fitxer principal:

```
/** ###################################################################
**     Filename  : Project_12.C
**     Project   : Project_12
**     Processor : MCF5213CAF80
**     Version   : Driver 01.00
**     Compiler  : CodeWarrior MCF C Compiler
**     Date/Time : 05/12/2009, 11:07
**     Abstract  :
**         Main module.
**         This module contains user's application code.
**     Settings  :
**     Contents  :
**         No public methods
**
** ###################################################################*/
/* MODULE Project_12 */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Cap1.h"
#include "Cap2.h"
#include "Cap3.h"
#include "AS1.h"
#include "ADC1.h"
#include "I2C1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

/* User includes (#include below this line is not maintained by Processor
void main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!!
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.
```

```
  /*** Don't write any code pass this line, or it will be deleted during c
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END Project_12 */
/*
** ###################################################################
**
**     This file was created by Processor Expert 1.04 [04.27]
**     for the Freescale MCF series of microcontrollers.
**
** ###################################################################
*/
```

Fitxer Events.c:

```
/** ###################################################################
**     Filename  : Events.C
**     Project   : Project_12
**     Processor : MCF5213CAF80
**     Beantype  : Events
**     Compiler  : CodeWarrior MCF C Compiler
**     Date/Time : 05/12/2009, 11:07
**     Abstract  :
**         This is user's event module.
**         Put your event handler code here.
**     Settings  :
**     Contents  :
**         Cpu_OnCoreWatchdogINT - void Cpu_OnCoreWatchdogINT(void);
**
** ###################################################################*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"

/* User includes (#include below this line is not maintained by Processor

/*
** ===================================================================
**     Event       :  Cpu_OnCoreWatchdogINT (module Events)
**
**     From bean   :  Cpu [MCF5213_100_LQFP]
**     Description :
**         This event is called when the OnCoreWatchdog interrupt had
**         occurred. This event is automatically enabled when the <Mode>
**         is set to 'Interrupt'.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cpu_OnCoreWatchdogINT(void)
{
  /* Write your code here ... */
}

uint32 ticks1;
uint32 ticks2;
uint32 ticks3;
double accel1;
double accel2;
double accel3;

unsigned short captura1 = 0;
unsigned short captura2 = 0;
```

```
unsigned short captura3 = 0;
unsigned short captura_temp1 = 0;
unsigned short captura_temp2 = 0;
unsigned short captura_temp3 = 0;
uint32 overflow1 = 0;
uint32 overflow2 = 0;
uint32 overflow3 = 0;
uint32 overflow_temp1 = 0;
uint32 overflow_temp2 = 0;
uint32 overflow_temp3 = 0;

int regtempA = 0;
int regtempB = 0;
double vtempA = 0;
double vtempB = 0;
double tA = 0;
double tB = 0;
double bA = -223.74;
double bB = -223.74;

double h[50] = {};
double R1[50] = {0};
double R2[50] = {0};
double R3[50] = {0};
double p1A = 2.121e-005;
double p2A = 0.002689;
double p3A = -0.01996;
double p1B = 2.121e-005;
double p2B = 0.002689;
double p3B = -0.01996;



double filtrar1(double mostra) {
    double s=0;
    int i;

    for(i=49; i>0; i--){
        R1[i]=R1[i-1];
        s=s+R1[i]*h[i];
    }

    R1[0]=mostra;
    return s+R1[0]*h[0];
}
```

```
double filtrar2(double mostra) {
    double s=0;
    int i;

    for(i=49; i>0; i--){
        R2[i]=R2[i-1];
        s=s+R2[i]*h[i];
    }

    R2[0]=mostra;
    return s+R2[0]*h[0];
}

double filtrar3(double mostra) {
    double s=0;
    int i;

    for(i=49; i>0; i--){
        R3[i]=R3[i-1];
        s=s+R3[i]*h[i];
    }

    R3[0]=mostra;
    return s+R3[0]*h[0];
}



/*
** ===================================================================
**     Event       :  Cap1_OnCapture (module Events)
**
**     From bean   :  Cap1 [Capture]
**     Description :
**         This event is called on capturing of Timer/Counter actual
**         value (only when the bean is enabled - <Enable> and the
**         events are enabled - <EnableEvent>.This event is available
**         only if a <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cap1_OnCapture(void)
{
    /* recollim les dades: temps i overflow */
    Cap1_GetCaptureValue(&captura1);
    overflow1 = overflow_temp1;
```

```
    if (testReg8Bit(GPTCTL2, EDG0A)) {      /* flanc de pujada */

        /* desactivem la interrupcio per flanc de pujada */
        clrReg8Bit(GPTCTL2, EDG0A);

        overflow_temp1 = 0;
        captura_temp1 = captura1;

        /* activem la interrupcio per flanc de baixada */
        setReg8Bit(GPTCTL2, EDG0B);

    } else {                                /* flanc de baixada */

        /* desactivem la interrupcio per flanc de baixada */
        clrReg8Bit(GPTCTL2, EDG0B);

        ticks1 = (overflow1 + captura1) - captura_temp1;
        accel1 = (ticks1*0.00002) - 4;

        /* obtenim la temperatura */
        regtempA = getReg16(ADRSLT0)>>4;
        vtempA = regtempA*0.001220703125;
        tA = vtempA*200 + bA;

        /* corregim les dades */
        accel1 = accel1*((tA*2.81)/8977600)-(p1A*tA*tA+p2A*tA+p3A);
        accel1 = filtrar1(accel1);

        /* activem la interrupcio per flanc de pujada */
        setReg8Bit(GPTCTL2, EDG0A);
    }
}

/*
** ===================================================================
**     Event       :  Cap2_OnCapture (module Events)
**
**     From bean   :  Cap2 [Capture]
**     Description :
**         This event is called on capturing of Timer/Counter actual
**         value (only when the bean is enabled - <Enable> and the
**         events are enabled - <EnableEvent>.This event is available
**         only if a <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
```

```
void Cap2_OnCapture(void)
{
    /* recollim les dades: temps i overflow */
    Cap2_GetCaptureValue(&captura2);
    overflow2 = overflow_temp2;

    if (testReg8Bit(GPTCTL2, EDG0A)) {      /* flanc de pujada */

        /* desactivem la interrupcio per flanc de pujada */
        clrReg8Bit(GPTCTL2, EDG0A);

        overflow_temp2 = 0;
        captura_temp2 = captura2;

        /* activem la interrupcio per flanc de baixada */
        setReg8Bit(GPTCTL2, EDG0B);

    } else {                                /* flanc de baixada */

        /* desactivem la interrupcio per flanc de baixada */
        clrReg8Bit(GPTCTL2, EDG0B);

        ticks2 = (overflow2 + captura2) - captura_temp2;
        accel2 = (ticks2*0.00002) - 4;

        /* obtenim la temperatura */
        regtempA = getReg16(ADRSLT0)>>4;
        vtempA = regtempA*0.001220703125;
        tA = vtempA*200 + bA;

        /* corregim les dades */
        accel2 = accel2*((tA*2.81)/8977600)-(p1A*tA*tA+p2A*tA+p3A);
        accel2 = filtrar2(accel2);

        /* activem la interrupcio per flanc de pujada */
        setReg8Bit(GPTCTL2, EDG0A);
    }
}

/*
** ===================================================================
**     Event       :  Cap3_OnCapture (module Events)
**
**     From bean   :  Cap3 [Capture]
**     Description :
**         This event is called on capturing of Timer/Counter actual
**         value (only when the bean is enabled - <Enable> and the
**         events are enabled - <EnableEvent>.This event is available
**         only if a <interrupt service/event> is enabled.
```

```
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cap3_OnCapture(void)
{
    /* recollim les dades: temps i overflow */
    Cap3_GetCaptureValue(&captura3);
    overflow3 = overflow_temp3;

    if (testReg8Bit(GPTCTL2, EDG0A)) {      /* flanc de pujada */

        /* desactivem la interrupcio per flanc de pujada */
        clrReg8Bit(GPTCTL2, EDG0A);

        overflow_temp3 = 0;
        captura_temp3 = captura3;

        /* activem la interrupcio per flanc de baixada */
        setReg8Bit(GPTCTL2, EDG0B);

    } else {                                /* flanc de baixada */

        /* desactivem la interrupcio per flanc de baixada */
        clrReg8Bit(GPTCTL2, EDG0B);

        ticks3 = (overflow3 + captura3) - captura_temp3;
        accel3 = (ticks3*0.00002) - 4;

        /* obtenim la temperatura */
        regtempB = getReg16(ADRSLT0)>>4;
        vtempB = regtempB*0.001220703125;
        tB = vtempB*200 + bB;

        /* corregim les dades */
        accel3 = accel3*((tB*2.81)/8977600)-(p1B*tB*tB+p2B*tB+p3B);
        accel3 = filtrar3(accel3);

        /* activem la interrupcio per flanc de pujada */
        setReg8Bit(GPTCTL2, EDG0A);
    }
}
```

```
/*
** ===================================================================
**     Event       :  Cap1_OnOverflow (module Events)
**
**     From bean   :  Cap1 [Capture]
**     Description :
**         This event is called if counter overflows (only when the
**         bean is enabled - <Enable> and the events are enabled -
**         <EnableEvent>.This event is available only if a <interrupt
**         service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cap1_OnOverflow(void)
{
    overflow_temp1 += 65536;
    overflow_temp2 += 65536;
    overflow_temp3 += 65536;
}


void aplicarConfiguracio(byte config){

}
/*
** ===================================================================
**     Interrupt handler : I2CRoutine
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__declspec(interrupt) void I2CRoutine(void)
{

    if(testReg8Bit(I2SR, IAAS) == 1){   /*hem estat adreçats*/

        if(testReg8Bit(I2SR, SRW) == 1){

            /* descomposem accel1 */
            byte *num1, *num2, *num3, *num4;
            num1 = ((byte*)(&accel1));
            num2 = ((byte*)(&accel1)+1);
            num3 = ((byte*)(&accel1)+2);
            num4 = ((byte*)(&accel1)+3);
```

```
/* enviem el primer */
setReg8(I2DR, num1);
while(testReg8Bit(I2SR, ICF) == 0);

/* enviem el segon */
setReg8(I2DR, num2);
while(testReg8Bit(I2SR, ICF) == 0);

/* enviem el tercer */
setReg8(I2DR, num3);
while(testReg8Bit(I2SR, ICF) == 0);

/* enviem el quart */
setReg8(I2DR, num4);
while(testReg8Bit(I2SR, ICF) == 0);

/* descomposem accel2 */
num1 = ((byte*)(&accel2));
num2 = ((byte*)(&accel2)+1);
num3 = ((byte*)(&accel2)+2);
num4 = ((byte*)(&accel2)+3);

/* enviem el primer */
setReg8(I2DR, num1);
while(testReg8Bit(I2SR, ICF) == 0);

/* enviem el segon */
setReg8(I2DR, num2);
while(testReg8Bit(I2SR, ICF) == 0);

/* enviem el tercer */
setReg8(I2DR, num3);
while(testReg8Bit(I2SR, ICF) == 0);

/* enviem el quart */
setReg8(I2DR, num4);
while(testReg8Bit(I2SR, ICF) == 0);

/* descomposem accel3 */
num1 = ((byte*)(&accel3));
num2 = ((byte*)(&accel3)+1);
num3 = ((byte*)(&accel3)+2);
num4 = ((byte*)(&accel3)+3);

/* enviem el primer */
setReg8(I2DR, num1);
while(testReg8Bit(I2SR, ICF) == 0);
```

```
            /* enviem el segon */
            setReg8(I2DR, num2);
            while(testReg8Bit(I2SR, ICF) == 0);

            /* enviem el tercer */
            setReg8(I2DR, num3);
            while(testReg8Bit(I2SR, ICF) == 0);

            /* enviem el quart */
            setReg8(I2DR, num4);
            while(testReg8Bit(I2SR, ICF) == 0);

        } else {
            aplicarConfiguracio(getReg16(I2DR));
        }

    }

    /* tornem a activar la interupcio */
    clrReg8Bit(I2SR, IIF);
}



/* END Events */

/*
** ###################################################################
**
**     This file was created by Processor Expert 1.04 [04.27]
**     for the Freescale MCF series of microcontrollers.
**
** ###################################################################
*/
```

Annex B. Calibrat d'un Sensor de Temperatura

En molts dispositius electrònics de gran diversitat trobem incorporat un sensor de temperatura per tal de realitzar correccions en la senyal, control de funcionament o qualsevol tipus d'adaptacions a l'ambient.

Aquests sensors analògics generen una senyal amb una tensió proporcional a la temperatura. Aquesta tensió, però, s'ha d'interpretar correctament, és a dir, establir l'equivalència entre el voltatge i la temperatura mesurada.

Si analitzem el funcionament d'aquests dispositius veurem com la correspondència entre la tensió i la temperatura és una recta amb una pendent i un terme independent:

$$y = mx + n \tag{1.1}$$

El nostre objectiu és trobar el valor d'aquestes dues constants.

Procediment a seguir

Per tal de conèixer el valor de la pendent, hem de consultar la documentació del dispositiu. En les especificacions del sensor de temperatura trobarem un valor que ens ha d'indicar el guany de tensió per cada grau, o bé els graus de guany de temperatura per cada V o mV.

En aquest moment només ens fa faltarà trobar el terme independent. Tenint la pendent ja correctament incorporada a la funció, hem de fer una captura simultània de la senyal del sensor i

de la temperatura a la que està el xip (amb un termòmetre extern). Aquests dos valors ens permetran aïllar el terme independent i trobar-ne el seu valor.

Per tal de capturar la senyal del sensor de temperatura, utilitzarem un equip de captura de dades. En el nostre cas utilitzarem la targeta de captura NI PCI-6024E de National Instruments i el programa MatLab per l'adquisició i tractament de les dades.

Amb la rutina del codi 17 capturarem tantes mostres com indiquem a la freqüència que escollim i les guardarem en un fitxer. Un cop les tinguem desades en farem la mitja, i considerarem que aquest valor de la tensió serà l'equivalent a la temperatura que ens indiqui el termòmetre extern del xip.

```
function temperatura(f, mostres, nom_fitxer)

%
% Captura mostres de la targeta de captura NI i les mostra
%
% f = frequencia de mostreig en Hz
% mostres = nombre de mostres a captura
% nom_fitxer = nom del fitxer on guardar les mostres
%

% Informacio de les targetes d'adquisicio instal·lades
%
%                                                    AdaptorDllName:
'C:\MATLAB\R2006b\toolbox\daq\daq\private\mwnidaq.dll'
%         AdaptorDllVersion: '2.9 (R2006b)'
%               AdaptorName: 'nidaq'
%                BoardNames: {'PCI-6024E'  'PCI-6024E'}
%         InstalledBoardIds: {'1'  '2'}
%     ObjectConstructorName: {2x3 cell}


ai = analoginput('nidaq',1);                % creem un objecte per a
input analògic
ai.InputType  =  'SingleEnded';                        %  'SingleEnded'
'Differential'  'NonReferencedSingleEnded'
addchannel(ai,[0]);                                % afegim canals al
analoginput
```

```
set(ai,'SampleRate',f);                        % freqüència de mostreig
sol·licitada
fmostreig = get(ai,'SampleRate');              % freqüència de mostreig
que retorna el sistema (si tot va bé, igual a la sol·licitada)

set(ai,'TriggerType','Immediate');          % tipus de trigger
set(ai,'SamplesPerTrigger',mostres);        % mostres abans del trigger

set(ai.Channel(1),'InputRange',[-5  5]);      %  rang  de  la  senyal
d'entrada

start(ai);                                  % iniciem la captura
[dades, temps] = getdata(ai);               % obtenim les dades

plot(dades), grid,                          % mostrem les dades
xlabel('mostres'),
ylabel('voltatge (V)');

save(nom_fitxer, 'fmostreig', 'dades');   % guardem les dades

delete(ai);                                 % eliminem ai
clear all;                                  % netejem tot l'entorn
```

Codi 17: Rutina captura temperatura

Annex C. Disseny d'un Protocol de Comunicació I^2C

L'objectiu d'aquest document és definir un protocol per a l'intercanvi de dades entre microcontroladors. Bàsicament es tracta de centralitzar totes les dades, és a dir, permetre que tots els xips enviïn les dades a un únic nucli que les guardarà. Per a fer-ho, utilitzarem un únic bus de dades i el protocol I^2C.

El protocol I^2C ens permet amb només un canal de dades i una senyal de rellotge comunicar diferents dispositius. En el cas d'utilitzar un bus, podem treballar fins amb 2^7 components I^2C diferents, donant lloc a unes molt bones possibilitats d'expansió.

I^2C es basa en dues grans figures: el mestre i l'esclau. Serà el mestre aquell dispositiu que prengui el control del bus i que decideixi quins dispositius hi tenen accés. L'esclau, en canvi, resta a l'espera de que un mestre l'activi i li enviï dades o bé li doni permís per a enviar-les.

Funcionament bàsic del Protocol I^2C

Essencialment, el protocol I2C permet transmetre *bytes* de dades, és a dir, enviar i rebre paquets de 8 bits. Per a fer-ho, es basa en un algorisme força senzill:

1. El mestre s'apodera de l'ús del canal posant-lo a zero.
2. El mestre envia un *byte* amb el identificador de l'esclau i un bit que indica si vol escriure en el canal o bé vol llegir de l'esclau.

3. L'esclau respon amb un ACK al paquet que ha rebut del mestre.
4. Si el mestre vol escriure en el canal, inicia la transmissió. Si per contra és l'esclau qui ha d'escriure, serà aquest qui comenci a enviar dades.
5. Després de cada paquet de dades enviat el receptor ha de confirmar la recepció correcta d'aquest mitjançant la posada a zero del canal.
6. Un cop el mestre ha acabat la transmissió o la recepció, pot continuar utilitzant el canal amb un altre esclau o bé pot enviar un STOP per alliberar el canal.

Disseny del Protocol Propi

Com ja hem dit, ens trobem davant d'un cas en que tenim un microprocessador que ha de recollir dades d'altres microcontroladors. Per tal de facilitar la implementació hardware del dispositiu, emprarem un únic bus per a la connexió entre tots els components.

A més a més, també volem que aquest bus de dades serveixi per a configurar els sistemes de captura de dades. Per això, si els microcontroladors responsables dels sensors reben un paquet de dades, entendran que es tracta de la configuració que han d'aplicar, en qualsevol moment.

En el nostre cas, el microcontrolador que recollirà totes les dades serà el mestre del bus, mentre que la resta de components I2C seran esclaus. L'algorisme de recollida de dades serà el següent:

1. El mestre prendrà el control del bus.

2. El mestre enviarà a cada un dels esclaus un *byte* amb la configuració que han d'adoptar.
3. El mestre recorrerà tots els esclaus un a un, enviant-los un missatge amb el seu ID i amb el bit de lectura actiu.
4. Cada esclau, al rebre aquest missatge, enviarà les dades de les que disposi, segons el format indicat per la configuració enviada en el pas 2.
5. Si l'esclau no rep el ACK corresponent, tornarà a enviar totes les dades.
6. Si el mestre no rep resposta d'un dels esclaus, provarà d'enviar-li de nou un missatge de configuració.
7. Si continua sense rebre resposta, passarà al següent esclau i notificarà l'error.
8. En qualsevol moment, el mestre pot enviar un missatge de configuració a un esclau, que l'ha de processar i adaptar-se a la nova configuració immediatament.

Els paquets de configuració seran *bytes* amb un significat concret. Cada esclau ha de tenir una taula de configuracions amb els valors adients de, per exemple, eixos a transmetre, ordre dels *bytes*, etc.
Aquesta taula s'ha de dissenyar per a cada tipus de sensor i s'ha de conèixer abans de poder finalitzar la programació del microcontrolador que el gestiona.

Annex D. Enviament de dades compostes a través d'I^2C

Un dels problemes amb els que ens hem hagut d'enfrontar al llarg de la realització d'aquest projecte ha sigut el fet d'haver d'enviar dades compostes a través d'I^2C, que només envia *bytes*.

Entenem dades compostes com aquelles variables que estan en un format major de un *byte*, i que per tant no es poden enviar directament a través d'aquest bus.

Per a enviar-les, les haurem de dividir en *bytes* i transmetre cada un d'aquests a través d'I2C. En el sistema destí, s'haurà de seguir el procediment invers per a tornar-les a muntar.

El procediment és força senzill. Crearem tants punters a byte com *bytes* ocupi la dada. En el nostre cas seran 4 ja que tractem amb *floats*. A aquests punters els assignarem l'adreça de cada un dels *bytes* que formen la variable.

En el cas de la primera posició de memòria no hi ha cap problema, però, i les consecutives? Doncs bé, el que farem és augmentar en una unitat l'adreça de memòria, i apuntarà al següent *byte* ja que tractem amb paraules de *byte*.

A tall d'exemple, hem generat un codi que implementa els dos procediments: el de desmuntar la variable en *bytes* i el de tornar-la a muntar.

```
void main(void)
{
    /* declarem 2 double i 4 punters a byte */
    double numero, numero2 = 0;
    byte *num1, *num2, *num3, *num4;

    /* fraccionem el double */
    num1 = ((byte*)(&numero));
    num2 = ((byte*)(&numero)+1);
    num3 = ((byte*)(&numero)+2);
    num4 = ((byte*)(&numero)+3);

    /* recomposem el double: num1.num2.num3.num4 */
    *((byte*)(&numero2)) = *num1;
    *((byte*)(&numero2)+1) = *num2;
    *((byte*)(&numero2)+2) = *num3;
    *((byte*)(&numero2)+3) = *num4;

}
```

Codi 18: Descomposició de variables compostes

El procediment per a recompondre la variable és força intuïtiu si s'ha entès el primer. Només consisteix en guardar els valors en posicions de memòria contigus i assignar a la variable composta l'adreça del primer *byte*.

L'ordre és molt important, per tant, emissor i receptor s'han de posar d'acord sobre si primer envien el *byte* de més pes o el de menys.

Annex E. Coeficients del Filtre

A continuació trobarem els coeficients del filtre que hem dissenyat amb MatLab:

```
%
% Generated by MATLAB(R) 7.7 and the Signal Processing Toolbox 6.10.
%
% Generated on: 11-Jan-2010 06:19:41
%

% Coefficient Format: Decimal

% Discrete-Time FIR Filter (real)
% -------------------------------
% Filter Structure  : Direct-Form FIR
% Filter Length     : 51
% Stable            : Yes
% Linear Phase      : Yes (Type 1)


Numerator:
-0.0010195589240706902
-0.0006525556372839103
 0.00040432153191682544
 0.0015468088385178417
 0.001680353854812416
-0.00000000000000000019648140327187824
-0.0027767065719181555
-0.0041544627504816961
-0.0016999377238366799
 0.0040280804388207461
 0.0084507029837106659
 0.0060704640416668551
-0.0038709819555722514
-0.014365173483556672
-0.014673617284836547
 0.00000000000000000079849638138702455
 0.021075203785727505
 0.029789729073946503
 0.01171935673280115
-0.027321176786334635
-0.058124277957155752
-0.044154938565382253
 0.031758716201015703
 0.14931951643540892
 0.25682768876447193
 0.3002848899152244
 0.25682768876447193
 0.14931951643540892
 0.031758716201015703
-0.044154938565382253
-0.058124277957155752
-0.027321176786334635
 0.01171935673280115
 0.029789729073946503
```

```
 0.021075203785727505
 0.00000000000000000079849638138702455
-0.014673617284836547
-0.014365173483556672
-0.0038709819555722514
 0.0060704640416668551
 0.0084507029837106659
 0.0040280804388207461
-0.0016999377238366799
-0.0041544627504816961
-0.0027767065719181555
-0.00000000000000000019648140327187824
 0.0016803538548124l6
 0.0015468088385178417
 0.00040432153191682544
-0.0006525556372839103
-0.0010195589240706902
```